НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

«На правах рукопису»
УДК 004.42

«До захисту допущено»

Завідувач кафедри
_____ І.Р. Пархомей
(підпис)
“___”_____2018 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 126 «Інформаційні системи та технології»

на тему: Дослідження методів візуалізації ациклічних орієнтованих графів у системах конструювання алгоритмів обчислень

Виконав: студент другого курсу, групи ІК-71мп
(шифр групи)

Бондарчук Андрій Васильович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Науковий керівник доцент, к.т.н., доцент Мелкумян К.Ю. __ _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2018 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Рівень вищої освіти – другий (магістерський)

Спеціальність 126 «Інформаційні системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ І.Р. Пархомей
(підпис)
«___»_____2018 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Бондарчуку Андрію Васильовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації «Дослідження методів візуалізації ациклічних орієнтованих графів у системах конструювання алгоритмів обчислень», _____
науковий керівник дисертації доцент, к.т.н., доцент Мелкумян К.Ю., _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «07» листопада 2018 р. № 4112-с

2. Термін подання студентом дисертації 12 грудня 2018

3. Об'єкт дослідження – бібліотеки візуалізації графів/діаграм обчислень.

4. Предмет дослідження – алгоритми візуалізації з'єднань графа.

5. Перелік завдань, які потрібно розробити – аналіз проблеми та існуючих рішень; аналіз і реалізація методу; розробка методу; дослідження ефективності розробленого методу.

6. Орієнтовний перелік ілюстративного матеріалу – п'ять плакатів

7. Орієнтовний перелік публікацій – одна публікація

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз предметної області	13.09.2018 р.	
2	Постановка задачі	15.09.2018 р.	
3	Аналіз інформаційного забезпечення	20.09.2018 р.	
5	Аналіз алгоритмічного забезпечення	25.09.2018 р.	
6	Розробка алгоритмічного забезпечення	15.10.2018 р.	
7	Розробка програмного забезпечення	01.11.2018 р.	
8	Маркетинговий аналіз стартап-проекту	10.11.2018 р.	
9	Висновки	15.11.2018 р.	

Студент _____ Бондарчук А.В.
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ Мелкумян К.Ю.
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

У роботі досліджено методи візуалізації ациклічних орієнтованих графів. Проведено аналіз існуючих методів відображення графа, наведено їх переваги та недоліки.

У розділі вибору технологій розробки розглянуто різні інструменти для розробки рівня даних, прикладного рівня та рівня представлення та зв’язку між ними. Також визначено архітектуру бібліотеки, розроблено та вимоги до системи.

Розроблено метод, який дозволяє оптимізувати швидкість знаходження найкоротших шляхів між вузлами направленого ациклічного графу. Даний метод може використовуватися іншими розробниками для відображення графа.

В розділі маркетингового аналізу стартап-проекту проаналізовано поточну ситуацію на ринку, розроблено стратегії та маркетингові плани для впровадження даного рішення.

Ключові слова: метод візуалізації графа, пошук найкоротших маршрутів, направлений ациклічний граф.

Розмір пояснювальної записки – 88 аркушів, містить 23 ілюстрації, 26 таблиць, 5 додатків.

ABSTRACT

Diploma project examines the problem of graph visualization in directed acyclic graphs. The analysis of existing graph visualization methods has been made. Author presented advantages and disadvantages of these methods.

The choosing developing tool section various tools section describes existing developing tools for level of data, application level and presentation level and communication between them. Also the architecture of library was defined, and database schema with system requirements are developed in this section.

The implemented method makes it possible to optimize speed of finding the shortest paths between nodes in a directed acyclic graph and improve efficiency of the graph visualization. The results can be used by other developers in their own purposes to improve their applications.

In the marketing analysis section, the current situation on the market was analyzed, strategies and marketing plans for the implementation of this solution were developed.

Keywords: graph visualization method, finding the shortest paths, directed acyclic graph.

Explanatory note size – 88 pages, contains 23 illustrations, 26 tables, 5 applications.

Пояснювальна записка
до магістерської дисертації

на тему: *Дослідження методів візуалізації ациклічних орієнтованих графів у системах конструювання алгоритмів обчислень*

Київ – 2018 року

Зміст

ВСТУП

Граф є поширеним методом для відображення зв’язків між певними даними. Причому він має широке застосування не тільки як інструмент вирішення задач, а й як основний засіб для побудови математичних та інформаційних моделей.

Зазвичай, корисно уявити граф в графічній формі, так щоб було видно його структуру. Це називається візуалізацією (зображенням) графа. Зображення графу – це двовимірна візуалізація ребер та вершин графа. Зазвичай, зображення графа повинно задовольняти параметрам зрозумілості, зручності читання, швидкості побудови зображення.

Можна навести декілька прикладів, де візуалізація графових структур є дуже корисною:

- ієрархії класів і пакетів вихідного коду будь-якої програми;
- візуалізація соціального графа (той же Twitter або Facebook);
- зображення графа цитувань (дослідження зв’язності статей, хто на кого посилаються).

Проте, розташування вершин і ребер графа не є простою задачею і може вимагати значної кількості витраченого часу в ручному режимі уже для графа із порядком 10 (порядок графа – кількість вершин). Саме тому візуалізація графів і алгоритмів на графах є актуальним завданням.

Одними з основних критеріїв ефективності алгоритму побудови зображення графа є:

- час візуалізації графа з великою кількістю вузлів і з’єднань (більше 500 вузлів і 1000 з’єднань) для ефективної взаємодії з користувачем;
- мінімальна кількість накладань ребер графа та мінімальна довжина з’єднань між вершинами графа.

На сьогодні існує безліч алгоритмів та бібліотек для візуалізації графа, проте дані рішення не відповідають вищевказаним критеріям. У зв’язку з цим пропонується розробити алгоритм візуалізації з'єднань графа який оптимізує візуалізації графів/діаграм обчислень.

Об’єкт дослідження: бібліотеки візуалізації графів/діаграм обчислень.

Предмет дослідження: алгоритми візуалізації з'єднань графа.

Методи: системний підхід для аналізу інформації, метод параметричної оптимізації для оптимізації алгоритму, структурно-функціональний метод для здійснення постановки задачі, Scrum підхід для розробки ПЗ та для управління проектом.

1. ПРОБЛЕМАТИКА

1.1. Дослідження сучасних методів програмування, визначення їх переваг та недоліків

Існує два основних підходи до написання програмного коду: текстове та візуальне програмування.

Під текстовим програмуванням будемо розуміти спосіб написання при якому код вводиться з клавіатури та зберігається у вигляді текстового файлу. Як правило, коли люди говорять про текстові мови програмування, то вони посилаються на такі мови програмування, як Python, C#, Java і JavaScript. Для написання програмного коду можна використовувати будь-який текстовий редактор. Проте, для підвищення ефективності, зазвичай, використовується інтегроване середовище розробки (IDE) програмного забезпечення. Підвищення продуктивні досягається завдяки візуально простим інтерфейсам в IDE та сильно зв’язаним компонентам, таким як текстовий редактор, компілятор тощо. Як правило, інтегроване середовище являє собою один програмний комплекс, в якому ведеться уся розробка. Він, зазвичай, має багато модулів для написання, редагування, компілювання, налагодження та розгортання програмного забезпечення. IDE дає можливість абстрагуватися від виконання другорядних задач, що дозволяє розробнику значно зменшити кількість витраченого часу на типові технічні дії (наприклад, виклик компілятора, запуск засобів для налагодження коду) та дає можливість зосередитися на виконанні бізнес-задач та алгоритмічних задач. Окрім цього, інтегроване середовище розробки дає можливість зробити аналіз написаного коду і миттєво сповістити розробника про синтаксичні помилки у коді. Це дозволяє значно зменшити час на розробку програм. Завдяки вищенаведеним можливостям збільшується продуктивність розробника.

Візуальне програмування – спосіб написання програми для електронно-обчислювальної машини методом графічного маніпулювання програмними елементами замість написання тексту [1]. Візуальне програмування часто

представляють як наступний етап розвитку текстових мов програмування. Наочним прикладом може служити Microsoft Visual Studio, де графічні об’єкти редагуються і водночас змінюється та показується відповідний програмний код. У зв'язку з розвитком мобільних пристроїв (смартфонів, планшетів) програмісти стали більше цікавитися візуальним програмуванням. Візуальне програмування в основному використовується для створення програм з графічним інтерфейсом для операційних систем з графічним інтерфейсом користувача.

Необхідно розрізняти наступні терміни:

– графічна мова програмування – мова, призначена для написання програми для комп'ютера або обчислювального пристрою, в якому замість текстового опису алгоритму роботи використовується графічний.
– візуальні засоби розробки – засоби проектування, в якій найбільш поширені блоки програмного коду представлені у вигляді графічних об'єктів. Застосовуються в основному для створення прикладних програм і розробки графічного інтерфейсу користувача (GUI).

Візуальні мови програмування можуть бути додатково класифіковані на: мови, що базуються на іконках; мови, що базуються на формах [1]. Візуальне середовище розробки забезпечує графічні елементи, які можуть інтерактивно управлятися користувачами відповідно до певної специфічної просторової граматики для побудови програми.

Загальна мета візуальних мов програмування – зробити програмування більш доступним для новачків та підтримувати програмістів на трьох різних рівнях [2]:

– Синтаксис. Візуальні мови програмування використовують іконки/блоки, форми та діаграми, які намагаються зменшити або навіть повною мірою уникнути можливість виникнення синтаксичних помилок.
– Семантика. Візуальні мови програмування можуть забезпечити деякі механізми для розкриття значення програмних примітивів. Вони можуть містити допоміжні функції, які забезпечують функції документації.

– Прагматика. Візуальні мови програмування мають можливість показати стан програми у конкретних ситуаціях. Це дозволяє користувачам розміщувати артефакти, створені за допомогою візуальної мови програмування, у певний стан, щоб вивчити, як програма реагує на цей стан. Приклади: в AgentSheets або AgentCubes користувачі можуть встановлювати гру чи симуляцію в певний стан, щоб побачити, як реагує програма. Користувач, що володіє мовою програмування Thymio, може помістити робота в певний стан, щоб побачити, як він буде реагувати, тобто, які датчики активуються.

Графічна мова програмування, як правило, використовує функцію перетягування, а не введення. Вона може містити іконки або текстові мітки на блоках і елементах. Часто використовуються такі GUI елементи, як діалоги та спадні меню вибору (drop-down menu).

Однією із важливих переваг візуальних мов програмування є те, що не потрібно запам’ятовувати великий список доступних команд та синтаксис. Це занадто велика кількість даних для дітей та початківців, які вивчають нову мову програмування.

Безліч професійних розробників програмного забезпечення витрачають десятки годин щотижня, програмуючи на одній мові. Це означає, що дорослі можуть запам'ятати багато про синтаксис і API.

Помилки в синтаксисі – величезна проблема дітей і дорослих. Наприклад, можна випадково не закрити дужку чи пропустити деякий символ. На вирішення цієї помилки можна витратити великий об’єм часу. Звичайно, хороші редактори можуть допомогти вам виявити ці помилки, але навіть найкращі редактори не дозволяють зробити синтаксичну помилку і не виправити її. У візуальних мовах ви працюєте безпосередньо з концепцією програмування замість того, щоб розбивати її на окремі фрагменти задач.

Зазвичай, у візуальному програмуванні блок-схема алгоритмів виконання відображується у вигляді графа, тому необхідно навести термінологію із теорії графів.

1.2.Ключові моменти теорії графів

У математиці теорія графів полягає у вивченні графів, які є математичними структурами, що використовуються для моделювання попарних зв'язків між об'єктами. Граф у цьому контексті складається з вершин(вузлів або точок), які з'єднані ребрами (дугами або лініями). Графи є одним з основних об'єктів дослідження в дискретній математиці.

У найбільш загальному розумінні граф – це упорядкована пара $G = (V, E)$, що складається з множини V вершин та множину E ребер. Причому множина ребер E є декартовим добутком $V \times V$ множини V. Це означає що кожно ребро з'єднує рівно дві вершини.

Теорія графів – розділ дискретної математики, що вивчає властивості графів. В загальному сенсі граф представляється як множина вершин (вузлів), з'єднаних ребрами $G = (V, E)$, де V – підмножина будь-якої множини, а E – підмножина $V \times V$.

Якщо на кожному ребрі обрати певний напрямок, то такий граф називається ***орієнтованим***, а його ребра з напрямками — ***дугами***. Для орієнтованого графа зберігаються всі поняття та характеристики, що відносяться до відповідного неорієнтованого.

Маршрутом між вершинами x_a та x_b у графі називають скінченну послідовність ребер та інцидентних їм вершин, що складають неперервну криву з кінцями x_a, x_b. Число ребер у маршруті називається його ***довжиною*** (включаючи внесок повторюваних ребер).

Маршрут називається ***ланцюгом***, якщо всі його ребра (дуги) різні, і ***простим ланцюгом***, якщо всі його вершини (крім кінців) різні.

Циклом у графі називається ланцюг, в якому перша та остання вершина збігається з початковою [3].

Топологічне сортування — впорядковування вершин безконтурного орієнтованого графа згідно з частковим порядком, визначеним ребрами цього графу на множині його вершин.

Розглянемо детальніше орієнтований ациклічний граф, адже даний клас графів широко використовується в інформаційних системах.

У математиці та інформатиці спрямованим ациклічним графом (directed acyclic graph - DAG) називають скінченний спрямованим граф без орієнтованих циклів. Тобто, він складається з кінцевої множини вершин і ребер, причому кожне ребро спрямовується з однієї вершини в іншу, так що немає можливості почати з будь-якої вершини v і слідувати послідовно спрямованій послідовності ребер, які зрештою зворотно повертаються до v [4]. Це означає що в даному графі не існує зв’язок як починається і закінчується в одній і тій же самій вершині. Орієнтований ациклічний граф є деяким узагальненням дерева. Точніше кажучи, він є об’єднанням дерев (лісом).

DAG можуть моделювати різні види інформації. Наприклад, електронну таблицю можна моделювати як DAG, де кожній комірці буде відповідати вершина графа, а формулі, яка в одній комірці використовує значення іншої буде відповідати ребро графа. Топологічне сортування цього DAG може використовуватися для оновлення всіх значень комірок при зміні таблиці. Аналогічним чином, топологічне сортування DAG може використовуватися для впорядкування операцій компілювання у make-файлі. Техніка оцінки та аналізу програм використовує DAG для моделювання етапів та діяльності проектів. Використання даного графа дозволяє запланувати виконання проекту так, щоб мінімізувати затрачений час на виконання проекту.

1.3. Дослідження проблем які виникають в задачах візуалізації графів

У системах візуального програмування блок-схема алгоритмів виконання відображується у вигляді орієнтованого направленого графа. Причому, розмір даного графа може бути дуже суттєвим. Зрозуміло, що для ефективної взаємодії з даним графом необхідно щоб він задовольняв основним візуально-естетичним параметрам [3]:

– *накладання*. Кількість накладань (схрещувань) в графі це число з'єднань, які візуально перетинаються. Існує домовленість що планарний (граф, який може бути зображений на площині без перетину ребер) граф рисується без накладань. Такий граф ще називають плоским. Проте, на практиці планарний граф практично не зустрічається. Зрозуміло, що при збільшенні ребер графа кількість потенційних перетинів збільшується, а якість і зрозумілість зображуваного графа – зменшується. Саме тому, мінімізація схрещувань є однією із головних задач при розробці методу візуалізації графа.
– *область зображення*. Область зображення – розмір найменшого обмежувального прямокутника, який будуються по відношенні до найменшої відстані між двома вершинами графа. Зазвичай, чим меншою є ця область, тим кращим є зображення графа, тому що це дозволяє показати більше вершин графа. Що, у свою чергу, дає можливість більш виразно показати структуру графа. Співвідношення сторін зображуваної області також є важливим параметром.
– *загальна/максимальна довжина ребер*. Декілька, часто вживаних параметрів якості зображення відносяться до довжини відрізків зображення. Зазвичай, бажано мінімізувати загальну довжину ребер графа. Це, у свою чергу, означає що контекстно-близькі вершини будуть знаходитися поряд (що також є важливим параметром оцінки якості візуалізації). Також, необхідно мінімізувати максимальну довжину усіх зв’язків. Крім того, для простішого сприймання краще щоб довжина відрізків була рівномірною.
– *загальна кількість згинів*. Для простоти отриманого зображення необхідно щоб з’єднання між вершинами графа являли собою якомога більш прості геометричні примітиви. Це робиться з метою спрощення сприйняття відрізків зображення оком людини. Зазвичай, складність з’єднання визначається кількістю згинів. Багато алгоритмів візуалізації графів

орієнтовані на мінімізацію згинів в цілому чи мінімізацію згинів для кожного з’єднання.

У системах візуального програмування кожна вершина може містити декілька входів-виходів. Відповідно, для зменшення складності отримуваного зображення графа необхідно реалізувати механізм візуалізації графів, оптимізуючи вищенаведені параметри.

Метою даної магістерської дисертації є оптимізація швидкості знаходження найкоротших шляхів між вузлами направленого графа з обходом вузлів з’єднаннями при візуальному відображенні графа.

1.4.Дослідження та опис алгоритмів візуалізації графів

Перед початком проектування та розробки певного алгоритму візуалізації графа необхідно провести аналіз вже існуючих рішень. Для кожного методу визначити недоліки та переваги, щоб мати змогу розробити власний алгоритм.

Існує багато алгоритмів для візуалізації графів. Розглянемо найпоширеніші з них, виділимо їх переваги та недоліки.

1.4.1. Алгоритм випадкового розміщення

Одним із найпростіших алгоритмів у візуалізації графів є алгоритм Випадкового (Random) розміщення. Суть даного методу полягає у тому що вершини розташовуються у довільному положенні та порядку. Вибирається прямокутна область потрібної величини. Далі у цій області випадковим чином розміщують всі вершини та ребра графа. Розподіл вершин у даній області є рівномірним.

Зрозуміло, що даний алгоритм дуже простий у реалізації та може візуалізувати граф із довільною кількістю вершин. Проте, даний метод не задовольняє параметрам естетичності та практичності, зокрема:

– отримане зображення не є симетричним. Даний метод не здійснює пошук груп симетрії в графі.

– даний метод навіть не намагається зменшити кількість накладань. Алгоритм не виконує жодних обчислень для уникнення зайвих накладань.
– ребра не мають просту форму.
– не забезпечується мінімальна відстань між контекстно-близькими вершинами. Мається на увазі що контекстно-близькі вершини можуть бути розташовані на різних кінцях зображення.
– не забезпечує візуальної картини представлення даних. З отриманого зображення практично нереально виділити якусь корисну інформацію, провести аналіз та зробити певні висновки.

Приклад візуалізації за допомогою алгоритму випадкового розміщення зображений на Рисунок 1.1.

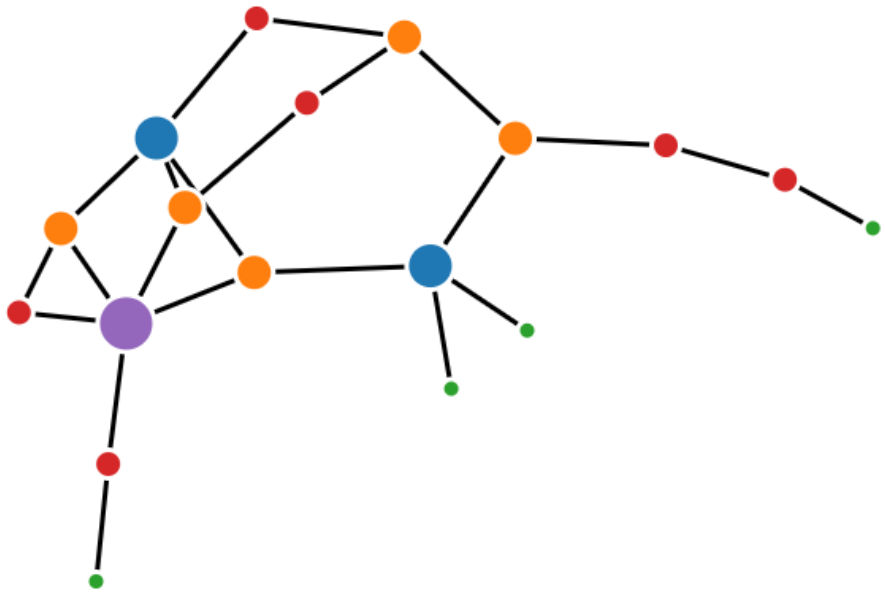


Рисунок 1.1 Візуалізація з допомогою алгоритму випадкового розміщення

1.4.2. Алгоритм кругового розміщення

Алгоритм кругового розміщення – алгоритм, в якому вершини графа розмішуються на колі. Зазвичай, відстані між вершинами графа є однаковими. Потрібно відмітити, що вказаний алгоритм практично не залежить від кількості вершин та ребер графа, а також дає хороший результат. Однією із переваг даного алгоритму є його нейтралітет. Це досягається тим що усі вершини

розташовуються на однакових відстанях один від одного та від центру круга. Тобто, жодній вершині не надається привілейована позиція, так як зазвичай користувач сприймає центрально розташовані вузли як важливіші. Вказана властивість кругового розміщення є корисною для деяких сфер, зокрема, вона активно використовується у біоінформатиці та аналізі соціальних мереж.

Ребра графа можуть бути зображені у вигляді хорд кола (Рисунок 1.2) чи дуг (Рисунок 1.3). Зображення з допомогою є хорд використовується лише при малій кількості вершин графа, адже із збільшенням вершин графа інформативність та зрозумілість отриманого зображення різко зменшується.

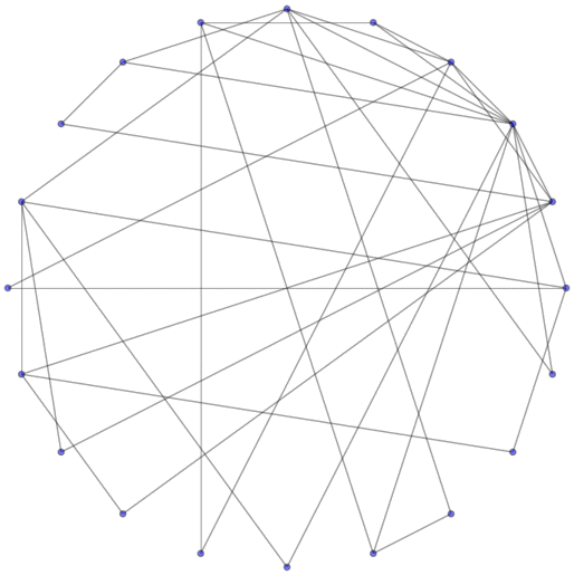


Рисунок 1.2. Зображення графа алгоритмом кругового розміщення з хордами

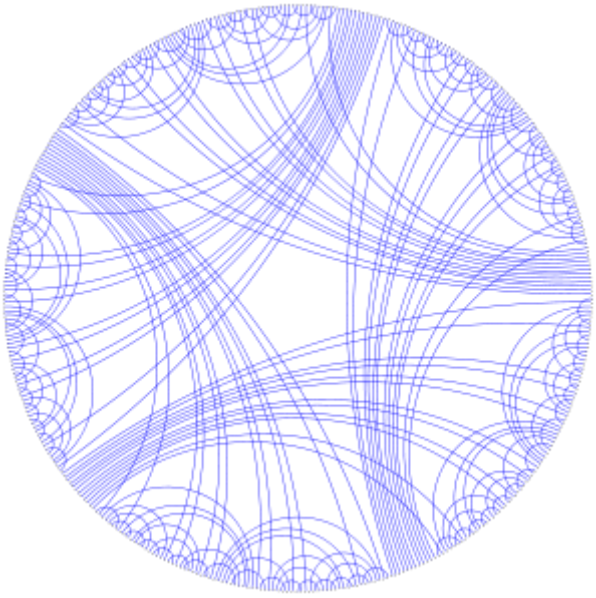


Рисунок 1.3 Зображення графа алгоритмом кругового розміщення з дугами

Задачу мінімізації кількості перетинів ребер графа в круговому розміщення вершин графа є NP-повною задачею. Для розв’язування цієї задачі використовується двоетапний алгоритм мінімізації кількості перетинів [5].

На першому етапі вершини додаються в один з кінців лінійного розміщення об’єктів. Вершина з максимальним степенем (максимальною кількістю ребр) вибирається першою. Далі здійснюється послідовний обхід усіх вершин алгоритмом обходу в ширину та послідовне додавання пройдених вершин в лінійне розміщення. Після додавання усіх вершин перший елемент з’єднується з останнім та отримується кругове розміщення вершин графа.

На другому етапі виконується мінімізації перетинів ребр. Для кожної вершини вибирається вершина, при перестановці з якою кількість перетинів максимально зменшується. Ця процедура продовжується до моменту, коли не можна буде знайти вершину, перестановка з якою дасть можливість зменшити кількість перетинів.

Попри всі свої переваги, круговий алгоритм візуалізації графа є незастосовним для масштабних і складний графів, адже він на надає інформацію про структуру графа.

1.4.3. Алгоритм дугового розміщення

Наступним алгоритмом візуалізації графа є алгоритм дугового розміщення (дугова діаграма). Дугова діаграма – це стиль подання графа, в якому вершини розташовані вздовж прямої на евклідовій площині, а ребра утворюються у вигляді півкіл на одній з двох плоских поверхонь [6]. Слід відмітити, що у деяких випадках відрізки прямої також використовуються для представлення ребер графа, якщо сусідні вершини графа містять зв’язок між собою (Рисунок 1.4).

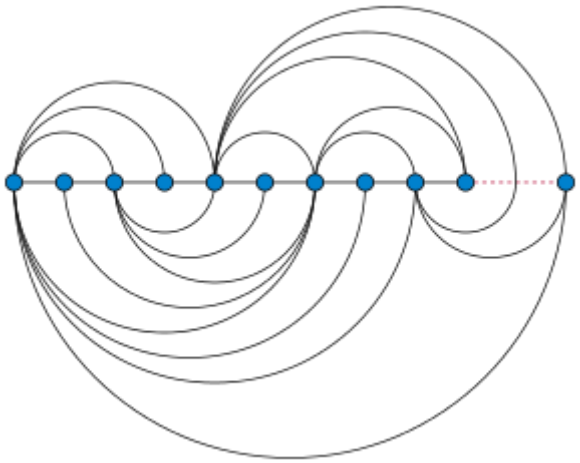


Рисунок 1.4. Зображення у вигляді дугової діаграми

Для зручності зображення орієнтованих графів існує домовленість напрямок зліва направо представляти у вигляді дуги над прямою, а напрямок справа наліво – під прямою. Ця домовленість називається орієнтацією по годинниковій стрілці і була розроблена як частина іншого стилю візуалізації графа, до складу якої входили Фекке, Ванг, Данг та Аріс. Пізніше, Преторіус і ван Вейк застосували це правило до дугових діаграм.

Дугові діаграми дають можливість відобразити багатомірні дані, які асоціюються з вершинами графа у простій формі. Проте, в даного методу представлення є один суттєвий недолік, сформульований авторами Хір, Босток и Огівєтскі. Вони писали, що дугові діаграми: «не можуть передати повну структуру графа так саме ефективно, як це робить двовимірне представлення» [7].

1.4.4. Алгоритм «Розміщення під дією сили»

«Розміщення під дією сили» (Force Directed Placement) – клас алгоритмів, спрямованих на візуалізацію графів із хаотично розміщеними вузлами, так щоб отримане в результаті зображення задовольняло параметрам естетичності (симетричності та мінімальності кількості перетинів ребр). Їх метою є розташування вершин графа в двовимірному чи тривимірному просторі так, щоб всі ребра були більш-менш однакової довжини та щоб кількість накладань була наскільки малою, наскільки це можливо(Рисунок 1.5). Це досягається присвоєнням вершинам та ребрам певної сили (зазвичай використовується закон Гука), яка відповідає відстані між вершинами. Мається на увазі, що чим більша відстань між вершинами, тим менша буде величина присвоєної сили і навпаки. На наступному етапі відбувається імітації руху ребер та вузлів. Ці рухи відбуваються в напрямку зменшення їхньої енергії [8].

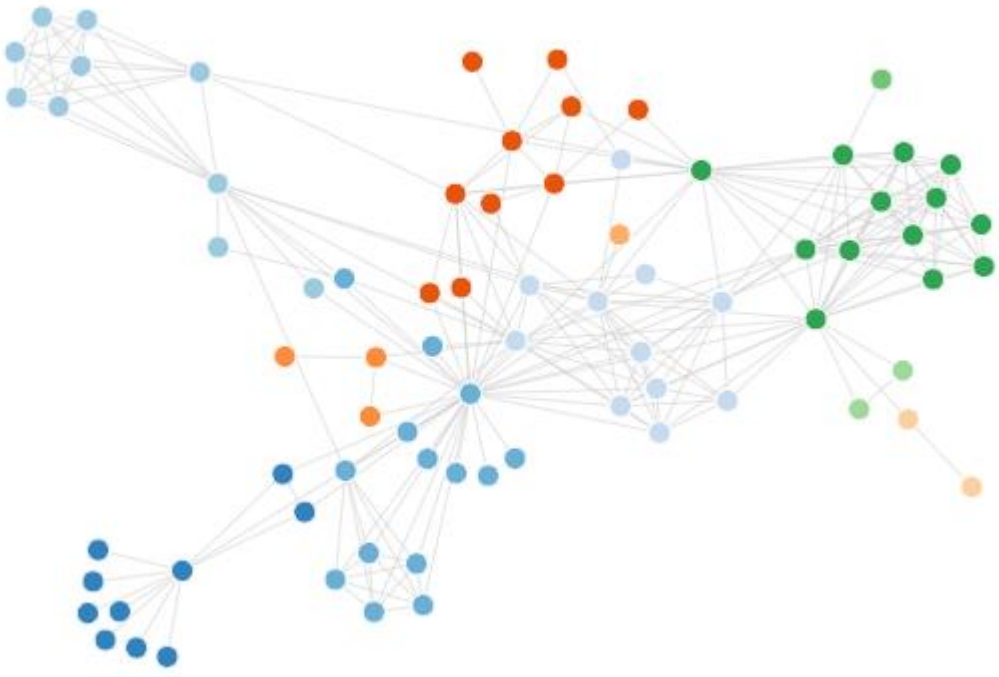


Рисунок 1.5. Візуалізація алгоритмом «Розміщення під дією сили»

Зображення графа може бути складною проблемою, проте алгоритми класу «Розміщення під дією сили», будучи фізичним моделюванням, зазвичай не вимагають спеціальних знань про теорію графів, таких як планарність.

Перелічимо переваги цього класу алгоритмів:

– Результати високої якості. Отримані результати мають, як правило, дуже хороші результати для графів середнього розміру (до 50-500 вершин). Зокрема, досягається рівномірність довжини ребра, рівномірного розподілу вершин та симетрію зображення. Симетричність є однією із найбільших переваг даного методу, адже її досить важко досягти з будь-яким іншим алгоритмом.
– Інтуїтивність. Даний алгоритм ґрунтуються на фізичних аналогіях добре відомих об’єктів, таких як, пружини. Завдяки цьому порівняно легко передбачити та зрозуміти поведінку даних алгоритмів.
– Простота. Зазвичай, алгоритми даного класу прості і можуть бути реалізовані в декількох рядках коду, що якісно відрізняє їх від інших алгоритмів.
– Інтерактивність. Іншою перевагою цього класу алгоритму є інтерактивний аспект. Виділяючи проміжні етапи побудови зображення, користувач може стежити за тим, як граф змінюється, бачачи, як він трансформується із заплутаної незрозумілої структури в привабливу конфігурацію. Існують інтерактивні програмні рішення, які дають можливість користувачу витягнути один або більше вузлів із стану рівноваги та спостерігати за тим, як вони повертаються в початкову позицію. Завдяки даній властивості даний клас алгоритмів використовується у динамічних системах та графічних онлайн системах.

Попри усі свої плюси, даний клас алгоритмів має декілька суттєвих недоліків, на які потрібно звернути увагу:

– Низька швидкодія при збільшенні кількості вершин графа. Типові алгоритми даного класу, загалом, вважаються такими, що мають алгоритмічну складність O(n3), де n - кількість вузлів вхідного графа. Це пояснюється тим, що кількість ітерацій оцінюється як O(n), і в кожній ітерації всі пари вузлів повинні бути відвідані для обчислення їхніх взаємно-відштовхувальних сил.
– Незадовільні локальні мінімуми. Даний клас алгоритмів створює графік із мінімальною енергією, зокрема, отриманий результат є лише локальним

мінімумом загальної енергії. Знайдений локальний мінімум в багатьох випадках може бути значно гіршим, ніж глобальний мінімум, що означає низьку якість отриманого зображення. Для багатьох алгоритмів із цього класу кінцевий результат може сильно залежати від початкового розташування вершин, яке в більшості випадків створюється випадковим чином. Зокрема, слід відмітити, що проблема локальних мінімумів стає більш помітною із збільшенням кількості вершин графа.

Одним із представників «Force Directed» алгоритмів є алгоритм Fruchterman-Reingold. Алгоритм Fruchterman-Reingold є корисним для візуалізації дуже великих не орієнтованих мереж. Він гарантує близькість вузлів, що логічно розміщені недалеко один від одного, і навпаки, віддаленість далеких вузлів. Алгоритм неможливо застосувати для мереж графових структур надто великих розмірів, через його низьку швидкодію.

Іншим представником «Force Directed» алгоритмів є алгоритм Kamada-Kawai Algorithm. Основна задача алгоритму залишається такою ж, як і у Fruchterman-Reingold алгоритмі. Основна відмінність алгоритму Kamada-Kawai полягає у мінімізації енергії шляхом пошуку похідної від рівнянь сили.

Алгоритм Kamada-Kawai виконує впорядкування вузлів графу досить швидко і тому може бути застосований для графів любих розмірів. Але, попри це, згідно загально-естетичних критеріїв результат, отриманий при використання цього алгоритму часто залишає бажати кращого, і тому досить часто після алгоритму Kamada-Kawai використовують Fruchterman-Reingold алгоритм [9].

1.4.5. Алгоритм Сугіями

Методика порівневого підходу була запропонована К. Сугіямою (K. Sugiyama). Методи порівневого представлення графів будують зображення, в яких вершини розподілені за рівнями. У зображеннях, спрямованих зверху вниз, рівні горизонтальні. Рідше будуються зображення, спрямовані зліва направо, з вертикальними рівнями. У разі горизонтальних рівнів, ребра представлені у

вигляді кривих, монотонно неспадаючих (висхідні зображення, upward) або незростаючих (спадні зображення, downward) у вертикальному напрямку.

Алгоритм Сугіями призначений перш за все для оптимізації візуальних-естетичних критеріїв, як зменшення довжин ребер та мінімізація перетинів ребр.

Вказаний алгоритм складається з трьох основних етапів (Рисунок 1.6):

– розподіл вершин за рівнями так, щоб напрямок дуг був незмінним (зверху вниз для низхідних, знизу вверх для висхідних зображень) і довжина кожного ребра була мінімальною.
– мінімізація кількості перетинів ребер за допомогою зміни порядку вершин на кожному рівні.
– вибір координат вершин на кожному рівні для мінімізації числа згинів ребр [10].

Для вирішення завдань кожного з зазначених етапів існує ряд евристик. Таким чином, конкретна реалізація є «конструктором» з різних методів, що вирішують завдання різних етапів.

Завдяки інтуїтивній зрозумілості та застосовності до широкого класу графів, порівневий підхід користується високою популярністю серед розробників систем візуалізації графів.

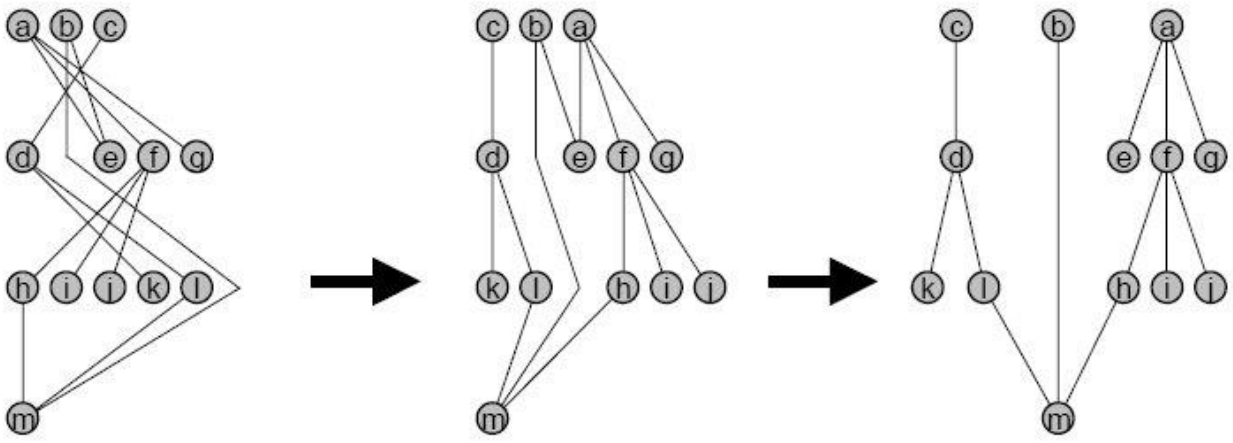


Рисунок 1.6**.** Ілюстрація алгоритму Сугіями

При всіх своїх перевагах у даного методу є декілька недоліків:

– алгоритм працює тільки для направлених графів;
– граф не повинен містити цикли.

1.4.6. Порівняння алгоритмів візуалізації графів

Було розглянуто декілька алгоритмів візуалізації графів. Визначимо параметри, які важливі при виборі оптимального алгоритму.

Алгоритм має задовольняти основним візуально-естетичним параметрам. Так, необхідно мінімізувати число з'єднань, які візуально накладаються. Це дозволить простіше слідувати від однієї вершини графа до наступної, що в свою чергу підвищить зрозумілість зображуваного графа.

Другим бажаним параметром є мінімізація довжини ребер отриманого графа. Це, також, також означає що пов'язані вершини будуть розміщені одна біля одної, що дозволить зрозуміти структуру графа із зображення.

Для забезпечення простоти зображення графа необхідно щоб з'єднання між вершинами мали просту геометричну форму. Зазвичай, складність такого з'єднання визначається кількістю згинів.

Останнім параметром є швидкість виконання алгоритму. Для ефективної взаємодії з користувачем необхідно щоб алгоритм був застосовним для складних і масштабних графів (більше 500 вузлів та 1000 з'єднань).

Відповідність алгоритмів даним вимогам можна знайти у табл. 1.1.

Таблиця 1.1. Порівняння алгоритмів візуалізації графів

Алгоритм / Параметр	Випадкове розміщення	Кругове розміщення	Дугова діаграма	Алгоритм «Розміщення під дією сили»	Алгоритм Сугіями
Мінімізація кількості накладань	–	+	+	+	+
Мінімізація довжини ребер	–	–	+	–	+

Продовження таблиці 1.1

Мінімізація кількості згинів	–	–	–	–	–
Застосовний для масштабних і складних графів	+	–	–	–	+

Як бачимо, серед розглянутих алгоритмів немає такого, який задовольнив би усі вимоги.

1.5. Дослідження та опис алгоритмів пошуку найкоротших маршрутів у графі

Задача про мінімізацію довжин ребер графа є надзвичайно актуальною для системи візуального програмування. Адже, як згадувалося раніше, мінімізація довжин з'єднань графа є корисним для спрощення та покращення естетичних показників зображення. Це дозволить розташувати сильно пов'язані вершини одна біля одної.

Задача про найкоротший шлях - завдання пошуку найкоротшого шляху між двома вершинами графа, в якій мінімізується сума ваг ребер, що складають шлях. Задача про знаходження найкоротшого шляху є однією із найбільш фундаментальних і типових задач теорії графів. Інакше вона ще називається задачею про мінімальний шлях. Практичне значення даної задачі визначається тим, що вона входить до складу інших більш складних задач та є моделлю для розробки ефективних методів вирішення задач теорії графів.

Значимість розв'язання даної задачі визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між двома перехрестями. Вершинами виступають перехрестя, а дороги – ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший.

Відзначимо, що на практиці з допомогою даної задачі можна розв’язувати не лише задачі обчислення довжини шляху, але й фінансові, матеріальні, паливно-енергетичні задачі. Так, наприклад, у економічних за своїм змістом задачах довжину ребра можуть відігравати такі величини, як час, вартість, обсяг витрачених ресурсів. Важливо, що при русі вздовж шляху ці характеристики додаються, відповідно, як і самі довжини при розрахунку загальної довжини. Це дає змогу мінімізувати вищевказані показники.

З теоретичної точки зору, цілком допустимо дослідження графів і з негативними значеннями ваг ребер. Однак задачі такого типу, як правило, не мають простого економічного змісту, вирішуються складніше Задача про найкоротший шлях має рішення для будь-якого зв'язного графа. При цьому під рішенням задачі розуміється сам найкоротший шлях і його довжина. Зрозуміло, що довжина найкоротшого шляху визначається однозначно, в той час як самих найкоротших шляхів може бути кілька. Рішення завдання про найкоротший шлях може бути проведено двома способами:

– на основі алгоритму повного перебору усіх можливих шляхів;
– методом динамічного програмування на графі.

Метод вирішення задачі про найкоротший шлях на основі алгоритму повного перебору шляхів, що з'єднують задані початкову і кінцеву вершини, є простим і очевидним. Спочатку будуються всі можливі шляхи між початковою і кінцевою точками. Обчислюються довжини усіх отриманих шляхів і вибирається мінімальна з них. Вибраний шлях (один або кілька) і буде розв’язком задачі. При цьому в сам алгоритм перебору шляхів можна внести незначні поправки, що дозволяють відкинути деякі явно невигідні варіанти. Наприклад, не потрібно продовжувати обчислення довжини шляху, якщо його довжина вже перевищує мінімальну довжину знайдених раніше шляхів. Зрозуміло, що основним недоліком даного алгоритму є його низька швидкодія та трудомісткість для складних графів. Тому, перейдемо до алгоритмів з використанням динамічного програмування.

Динамічне програмування - це математичний метод оптимізації, який знайшов застосування в багатьох галузях, від аерокосмічної промисловості до економіки. В основі даного методу лежить ідея спрощення складної задачі, розбиваючи її на більш прості підзадачі. Даний підхід є дуже корисним в ситуації коли деякі підзадачі виконуються дуже багато разів, адже кожна маленька задача виконується тільки один раз, що в свою чергу скорочує кількість потенційних повторних обчислень.

Якщо проблема може бути вирішена оптимально, розбиваючи її на під проблеми, а потім рекурсивно шукаючи оптимальні рішення підзадач, то вона, як кажуть, має оптимальну структуру.

1.5.1. Алгоритм Дейкстри

Одним із прикладів використання принципів динамічного програмування є алгоритм Дейкстри для пошуку найкоротшого шляху. Алгоритм знаходить найкоротший шлях між вказаною вершиною графа та усіма іншими вершинами графа. Також, можна знайти відстань між деякою початковою і кінцевою вершиною. Для цього виконання алгоритму починається із вказаної точки і продовжується до моменту, поки не буде досягнута очікувана кінцева точка. Наприклад, якщо вершини графа будуть представлені містами, а ребра – дистанцію між цими містами, то алгоритм Дейкстри може бути використаний для знаходження мінімальної відстані між деяким початковим містом і всіма іншими. Завдяки наведеній властивості, даний алгоритм широко використовується в мережевих протоколах маршрутизації (таких як IS-IS і OSPF) [11].

Наведемо опис послідовності виконання алгоритму Дейкстри:

– Усі вершини графа позначаються як ”невідвідані”. Алгоритм виконується для кожної точки лише один раз і для уникнення повторного виконання вводиться термін ”відвідана” вершина (Рисунок 1.7).

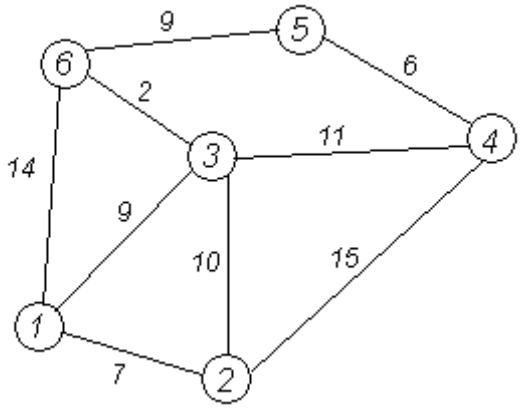


Рисунок 1.7. Усі вершини ‘невідвідані’

– Кожній вершині графа присвоюється проміжне значення довжини відрізка. Початковій вершині присвоюється значення 0, усім іншим – безкінечність (Рисунок 1.8). Початкова вершина позначається як поточна вершина.

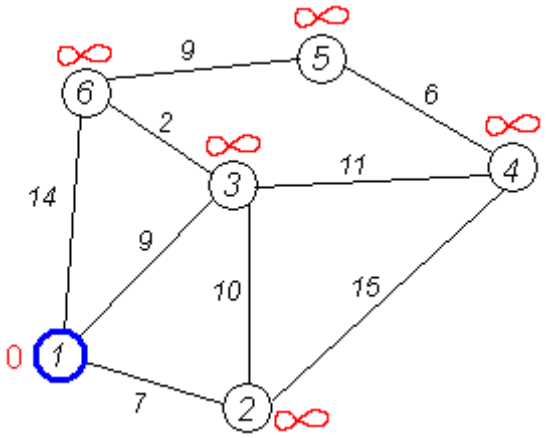


Рисунок 1.8. Початкова ініціалізація значень

– Знаходять сусідні вершини для поточної вершини і обчислюється відстань від поточної вершини до усіх сусідніх. Якщо отримане розрахункове значення довжини менше за значення мітки сусіда, то мітка сусіда замінюється на отримане значення довжини (Рисунок 1.9). Наприклад, якщо поточна вершина A позначена відстанню 6, а ребро, що з'єднує його з сусідом B, має довжину 2, тоді відстань до B через A буде дорівнювати 6 + 2 = 8. Якщо вершина B була позначена міткою, більшою за 8, то значення мітки замінюється на 8. В іншому випадку значення мітки не змінюється.

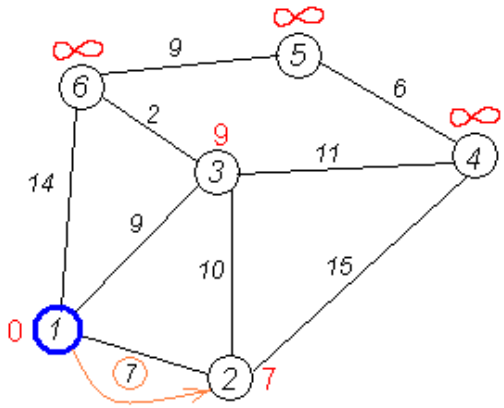


Рисунок 1.9. Визначення вершини з мінімальною вагою

– Після закінчення розглядання усіх сусідів поточної вершини, поточна вершина позначається ”відвіданою” (Рисунок 1.10). ”Відвідана” вершина більше не буде перевірятися у наступних кроках.

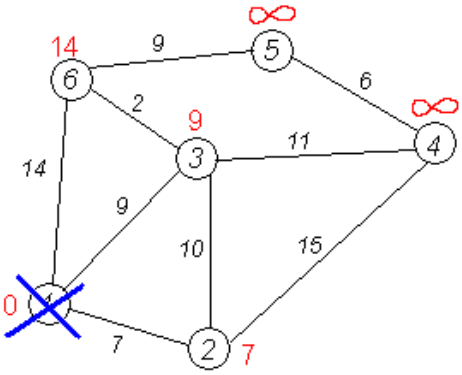


Рисунок 1.10. Позначення ‘відвіданої’ вершини

– Алгоритм повторюється до тих пір поки усі вершини не будуть ”відвіданими” (Рисунок 1.11)

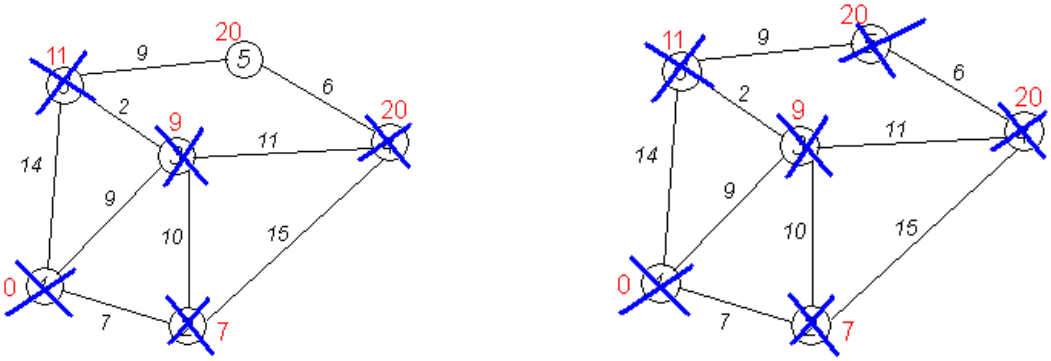


Рисунок 1.11. Закінчення виконання алгоритму Дейкстри

При обчисленні маршруту не обов’язково чекати поки кінцева вершина буде позначена ”відвіданою”: алгоритм може бути завершеним у випадку коли

кінцева вершина має найменшу розраховану дистанції серед усіх ”невідвіданих” вершин. У деякий випадках це може значно скоротити час виконання алгоритму.

1.5.2. Алгоритм пошуку A*

Пошук A* – в інформатиці та математиці, алгоритм пошуку по першому найкращому збігу на графі, який знаходить найкоротший маршрут від однієї вершини (початкової) до іншої (кінцевої).

Порядок обходу вершин визначається евристичною функцією «відстань + вартість» (зазвичай позначається як f (x)). Ця функція – сума двох інших: функції вартості досягнення розглянутої вершини (x) з початкової вершини (зазвичай позначається як g (x) і може бути як евристичної, так і не), а також функції евристичної оцінки відстані від поточної вершини до кінцевої (позначається як h (x)) [12].

Функція h (x) повинна бути допустимою евристичною оцінкою, тобто не повинна переоцінювати відстані до цільової вершини. Наприклад, для задачі маршрутизації h (x) може представляти собою відстань до цілі по прямій лінії, так як це фізично найменша можлива відстань між двома точками.

Цей алгоритм був вперше описаний в 1968 році Пітером Хартом, Нілсом Нільсоном і Бертрамом Рафаелем. По суті це було розширення алгоритму Дейкстри. Новий алгоритм досяг більш високої продуктивності (критерієм часу) за допомогою евристики. В їх роботі він згадується як «алгоритм A». Але так як він обчислює найкращий маршрут для заданої евристики, то він був названий A*.

A * покроково переглядає всі маршрути, що ведуть від початкової вершини в кінцеву, поки не знайде мінімальний. Як і всі евристичні алгоритми пошуку, він переглядає спочатку ті маршрути, які «здаються» ведуть до мети. Від жадібного алгоритму, який теж є алгоритмом пошуку по першому найкращому збігу, його відрізняє те, що при виборі вершини він враховує, крім іншого, весь пройдений до неї шлях. Складова g (x) - це вартість шляху від початкової вершини, а не від попередньої, як в жадібному алгоритмі.

На початку роботи проглядаються вузли, суміжні з початковим; вибирається той з них, який має мінімальне значення f (x), після чого цей вузол розкривається. На кожному етапі алгоритм оперує з безліччю шляхів з початкової точки до всіх ще не розкритих (листових) вершин графа - безліччю приватних рішень, - яке розміщується в черзі з пріоритетом. Пріоритет шляху визначається за значенням f (x) = g (x) + h (x). Алгоритм продовжує свою роботу до тих пір, поки значення f (x) цільової вершини не виявиться меншим, ніж будь-яке значення в черзі, або поки все дерево не буде переглянуте. З безлічі рішень вибирається рішення з найменшою вартістю.

Чим менше евристика h(x), тим більший пріоритет має вершина, тому для реалізації черги можна використовувати сортувальні дерева.

Безліч переглянутих вершин зберігається в closed, а які потребують розгляду шляху - в черзі з пріоритетом open. Пріоритет шляху обчислюється за допомогою функції f (x) всередині реалізації черги з пріоритетом.

Безліч вже пройдених вершин можна опустити (при цьому алгоритм перетвориться в пошук по дереву рішень), якщо відомо, що рішення існує або якщо successors вміє відсікати цикли.

Алгоритм A* допустимий і обходить при цьому мінімальну кількість вершин, завдяки тому, що він працює з «оптимістичною» оцінкою шляху через вершину. Оптимістичною в тому сенсі, що, якщо він піде через цю вершину, у алгоритму «є шанс», що реальна вартість результату буде дорівнює цій оцінці, але ніяк не менше. Але, оскільки A* є поінформованим алгоритмом, така рівність може бути цілком можливою.

Коли A* завершує пошук, він, згідно з визначенням, знайшов шлях, справжня вартість якого менше, ніж оцінка вартості будь-якого шляху через будь-який відкритий вузол. Але оскільки ці оцінки є оптимістичними, відповідні вузли можна без сумнівів відкинути. Інакше кажучи, A* ніколи не пропустить можливості мінімізувати довжину шляху і тому є допустимим.

Припустимо тепер, що якийсь алгоритм B повернув в якості результату шлях, довжина якого більше оцінки вартості шляху через деяку вершину. На

підставі евристичної інформації, для алгоритму B не можна виключити можливість, що цей шлях мав і меншу реальну довжину, ніж результат. Відповідно, поки алгоритм B переглянув менше вершин, ніж A*, він не буде допустимим. Отже, A* проходить найменшу кількість вершин графа серед допустимих алгоритмів, що використовують таку ж точну (або менш точну) евристику.

1.6.Постановка задачі дослідження

В результаті проведеного аналізу предметної області та існуючих аналогів було визначено мету магістерської дисертації – оптимізувати швидкість знаходження найкоротших шляхів між вузлами направленого графа з обходом вузлів з'єднаннями при візуальному відображенні графа. Для виконання поставленої мети необхідно розробити метод візуалізації графових структур.

Для досягнення мети магістерської дисертації необхідно вирішити наступні задачі:

- зменшити час візуалізації графа з великою кількістю вузлів і з'єднань (більше 500 вузлів і 1000 з'єднань) для ефективної взаємодії з користувачем
- прокласти з'єднання між вузлами графа мінімізувавши довжини шляхів та кількість накладань
- довести ефективність запропонованого методу

При розробці нового алгоритму необхідно визначити критерії оптимальності, тобто деякі показники, при яких можна сказати що алгоритм працює правильно і задовольняє потреби. Критерій оптимальності – характерний показник розв'язку задачі, по значенню якого оцінюється оптимальність знайденого розв'язку (максимальна наближеність до поставлених вимог). Правильний вибір критерії оптимальності відіграє суттєву роль в виборі оптимального вирішення поставлених задач.

В даному випадку слід виокремити наступні критерії оптимальності:

- мінімальна зважена довжина з'єднань графа між його вузлами;

– з’єднання розбиті на ортогональні відрізки, які повинні розташовуватись таким чином щоб уникати перетину з вузлами графа на його візуальному відображенні.

Крім цього, необхідно реалізувати бібліотеку візуалізації графів, щоб наглядно на прикладах показати виконання поставлених задач та задоволення критеріїв оптимальності.

1.7.Використані методи досліджень

Для аналізу інформації був використаний системний підхід. Мається на увазі, що об’єкт дослідження, бібліотека візуалізації графів/діаграм обчислень, розглядається як комплекс взаємопов’язаних компонентів. Системний підхід виявляє взаємозв’язки та закономірності в об’єкті з метою їх більш ефективного використання. Також необхідно зазначити, що системний підхід не є засобом для вирішення деяких задач, а є методом для постановки задач. Основним принципом системного підходу є цілісність, яка дає змогу аналізувати об’єкт як одне ціле та водночас, як підсистему для вищих рівнів. Наступним вагомим принципом є структуризація, яка дає можливість аналізувати елементи системи і їх взаємозв’язки в рамках конкретної організаційної структури.

В даній магістерській дисертації для постановки задачі використаний структурно-функціональний метод. Структурно-функціональний метод – підхід при поясненні та описі системи, який орієнтується на виявлення структури системи. Під структурою системи розуміється комплекс взаємозв’язків між елементами системи та їх ролі. Також необхідно зазначити, що структурно-функціональний метод є різновидом системного підходу.

Даний підхід вимагає:

– вивчення структури об'єкта;
– аналіз елементів досліджуваного об’єкта і їх функціональних характеристик;
– дослідження зміни елементів та їх функцій;
– розгляд розвитку об'єкта в цілому.

Для оптимізації алгоритму було використано метод параметричної оптимізації. Параметрична оптимізація – процес знаходження значень параметрів, при яких досягається максимальна ефективність виконання процедури або забезпечуються найкращі характеристики пристрою або системи з точки зору встановленого критерію. За кількістю параметрів, які оптимізуються, задача оптимізації може бути однопараметричною чи багатопараметричною, за кількістю критеріїв – однокритеріальною чи багатокритеріальною. Виходячи із поставлених задач до магістерської дисертації та сформульованих критеріїв оптимальності, можна зробити висновок, що задача є багатопараметричною та багатокритеріальною.

Для розробки ПЗ та управління використаний Scrum підхід. Scrum підхід є однією із методологій Agile. Agile визначають як набір підходів для «гнучкої» розробки ПЗ. Під «гнучкою» розуміється розробка, яка проводиться короткими циклами(ітераціями) тривалістю 1–4 тижні. Зведення розробки до серії коротких ітерацій сприяє мінімізації ризиків. Хоча, зазвичай, одного циклу недостатньо для випуску нової версії продукту, але мається на увазі те, що замовник отримує цінний для нього додаток(чи його частину), яке можна використовувати. Команда розробників постійно працює із замовником у ході проекту. Методологія Agile дає можливість замовнику змінювати вимоги під час виконання проекту, оскільки одна з основних ідей Agile визначає, що реакція на зміни важливіша, ніж дотримання початкового плану.

Scrum – набір методів, на яких побудований процес розробки, який дозволяє в фіксовані та невеликі за часом проміжки(від 1 тижня до місяця) надати користувача ПЗ із новими можливостями, для яких визначений найвищий пріоритет. В Scrum підході розрізняють три основні ролі: Власник продукту, Scrum-майстер, команда розробників. Власник продукту представляє інтереси кінцевих користувачів та відповідає за створення вимог до продукту. Scrum-майстер відповідальний за координацію роботи команди та підтримання атмосфери в команді. Команда розробників реалізує поставлені вимоги.

Набір можливостей, котрі реалізуються упродовж кожного циклу, надходять з етапу product backlog. Протягом одного спринту команда розробників виконує визначений список завдань. Упродовж цього періоду ніхто не може вносити зміни в список завдань. Також кожного дня відбувається зустріч розробників та Scrum-майстера, на якій команда обмінюється досвідом та стає зрозуміло хто і над якими задачами буде працювати сьогодні. Якщо у когось із команди виникають проблеми, які заважають досяганню цілі циклу, то Scrum-майстер допомагає у їх вирішенні.

1.8. Висновки

У даному розділі було проведено порівняльний аналіз текстового та візуального програмування, наведено базову термінологію теорії графів. Було досліджено проблеми, які виникають при візуалізації графів. На основі цих проблем було визначено мету магістерської дисертації та здійснено постановку задач. Також було проведено аналіз існуючих алгоритмів візуалізації графів. Було розглянуто особливості кожного алгоритму, наведено їх переваги та недоліки. На основі проведеного аналізу сформована порівняльна таблиця та зроблено висновок про те що існуючі алгоритми не задовольняють сформульованим критеріям оптимальності. Також, у даному розділі було проведено дослідження та опис алгоритмів пошуку найкоротших шляхів у графі. Поставлено задачі, основною метою яких є створення алгоритму візуалізації з'єднань графа, який буде виконувати поставлену мету та задовольнятиме критеріям оптимальності.

2. МОДИФІКОВАНИЙ МЕТОД ЗНАХОДЖЕННЯ НАЙКОРОТШИХ ШЛЯХІВ ТА ПРОКЛАДАННЯ З'ЄДНАНЬ У ОРІЄНТОВАНОМУ АЦИКЛІЧНОМУ ГРАФІ

В попередньому розділі було сформульовано задачу по оптимізації швидкість знаходження найкоротших шляхів між вузлами направленого графа з обходом вузлів з'єднаннями при візуальному відображенні графа. Для цього пропонується метод, який буде описаний у наступному пункті.

2.1.Опис методу

Задачу візуалізації графа, умовно, можна розділити на два етапи. На першому етапі відбуватиметься певні кроки для просторового розміщення вершин, на другому – позиціонування зв'язків.

При розташуванні вершин графа необхідно враховувати що вершини повинні розташовуються таким чином, щоб задовольняти двом естетичним параметрам: близькості пов'язаних вершин і зменшенні області зображення.

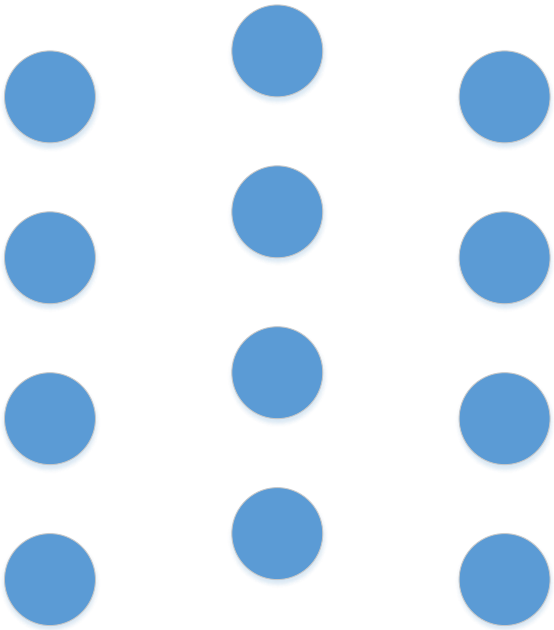


Рисунок 2.1. Просторовий розподіл вершин графа

Так як розглядається орієнтований ациклічний граф, це дає можливість візуалізувати граф без зворотних зв'язків. Даний параметр також є корисним, адже він суттєво підвищує зрозумілість отриманого зображення графа. Звичайно, що деякі алгоритми не намагаються побудувати граф без зворотних зв'язків.

Саме тому на цьому кроці необхідно використати алгоритм, котрий надає можливість побудувати граф без зворотних зв’язків. Одними із таких алгоритмів є алгоритми порівневого представлення графів, де вершини розподілені за рівнями. Саме тому на даному кроці методу пропонується використати один із цих алгоритмів, наприклад, алгоритм Сугіями.

В результаті вищеописаного кроку буде отримано просторовий розподіл вузлів графа згідно їх ступеня віддаленості від вхідних портів, що продемонстровано на рис. 2.2.

Після того як вершини розташовані у просторі, можна перейти до другого етапу методу: позиціонування зв’язків. Необхідно зобразити зв’язки між вершинами. При цьому необхідно пам’ятати що лінії повинні не накладатися, не перетинатися та містити мінімальну кількість згинів. Існує декілька основних варіантів для відображення зв’язків, такі як: лінія, ламана чи крива.

З’єднання у вигляді прямої лінії має суттєвий недолік: при деякому розташуванні вершин графа існує ймовірність того що лінія перетинатиме вершини (Рисунок 2.2). Це в свою чергу зменшує зрозумілість отриманого зображення графа та може призвести до неправильного трактування (розуміння) структури графа. Також, при даному типі ліній дуже важко уникнути перетини ребер, що не задовольняє відповідний параметр естетичності графа.

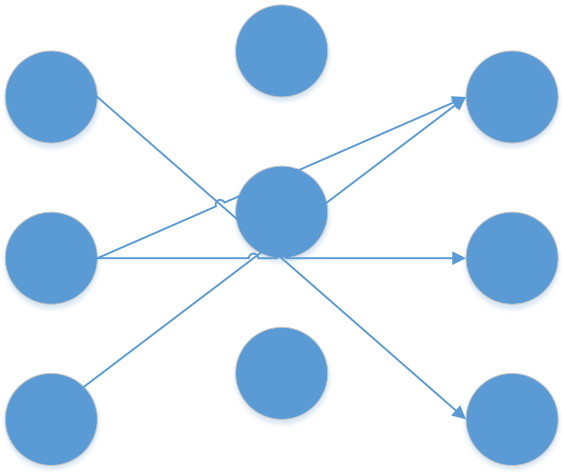


Рисунок 2.2. Візуалізація зв’язків за допомогою прямих ліній

Для даного методу було обрано візуалізацію відрізків у вигляді ортогональних відрізків, що дозволить легко слідувати від одної вершини до

іншої та забезпечить простоту отриманого зображення. Для побудови ортогональних відрізків вводиться допоміжна мережі “шин”, яка буде виступати прототипом для відрізків. Дана мережа будується між горизонтальними рядами вершин, а також між кожною парою вертикальних вершин (зображено на Рисунок 2.3). Точки, в яких відрізки перетинаються між собою є точками віртуального графа та будуть використані далі для прокладання відрізку. Відрізки між вершинами віртуального графа будуть складовими частинами результуючого з’єднання між вершинами графа. Зрозуміло, що перпендикулярність з’єднань віртуального графа забезпечить ортогональність окремих відрізків з’єднань основного графа.

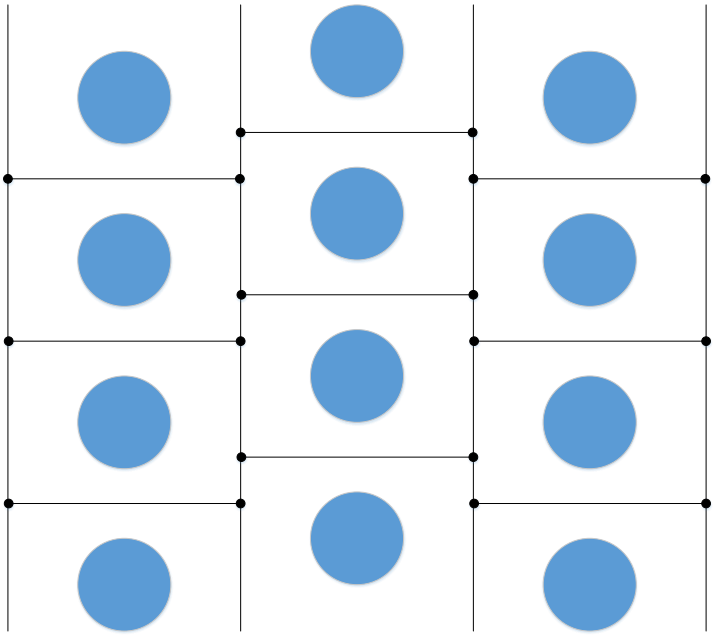


Рисунок 2.3. Допоміжна мережа “шин”

Зрозуміло, що між від деякої початкової точки до кінцевої може бути безліч різних шляхів через допоміжну мережу “шин”. Проте, необхідно знайти мінімальну відстань між цими точками. Це можна виконати з допомогою алгоритмів пошуку найкоротшого шляху.

В алгоритмах пошуку найкоротшого шляху мінімізується сума ваг ребр, які складають маршрут. Тому, необхідно підібрати вагові коефіцієнти таким чином, щоб задовільнити естетичним параметрам візуалізації графа. Основною складовою ваги для відрізка є довжина відрізка. Проте існують інші параметри,

які додатково збільшують вагу відрізка. Зокрема, будемо збільшувати вагу відрізку в наступних ситуаціях:

– при накладанні відрізків різних з'єднань один на одне
– при зміні напрямку наступного відрізка з'єднання з горизонтального на вертикальний. Даний коефіцієнт дозволить мінімізувати кількість поворотів (згинів) між відрізками з'єднання.
– при прокладанні відрізка з'єднання в зворотному напряму щодо загального напряму графа. Орієнтований ациклічний граф завжди може бути зображений без зворотних зв'язків, тому даний коефіцієнт є дуже важливим.

Відзначимо, що вага з'єднання між двома вершинами буде визначатися як суму ваг відрізків цього з'єднання. В результаті виконання цього кроку отримаємо зображення, як на Рисунок 2.4. З рисунка помітно що у даного кроці алгоритму є недолік, адже деякі з'єднання накладаються одне на одне, через що структура графа є незрозумілою. Для усунення цього недоліку необхідно розгорнути шини у з'єднання, що не накладаються одне на одне.

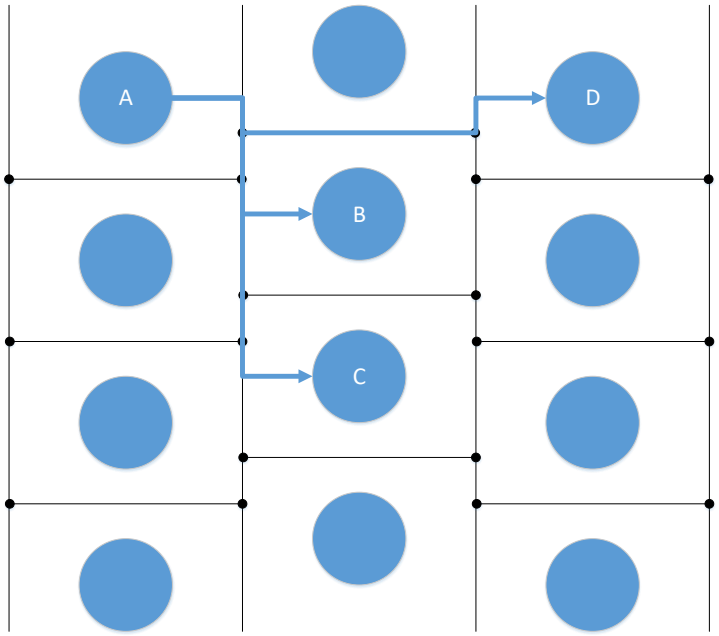


Рисунок 2.4. Граф після виконання алгоритму Дейкстри

Розглянемо Рисунок 2.4 більш детально. Із вершини А виходять три зв'язки і на рисунку всі виходи зображені однаково. Ці виходи можна рознести між

собою. Проте, потрібно зробити це “правильно” (як на Рисунок 2.5), щоб уникнути небажаних перетинів (як на Рисунок 2.6). Помітимо, що при “правильному” розташуванні відрізків першим із вершини A виходить відрізок із найменшою сумарною вагою (довжиною) та котрий повертає ліворуч першим.

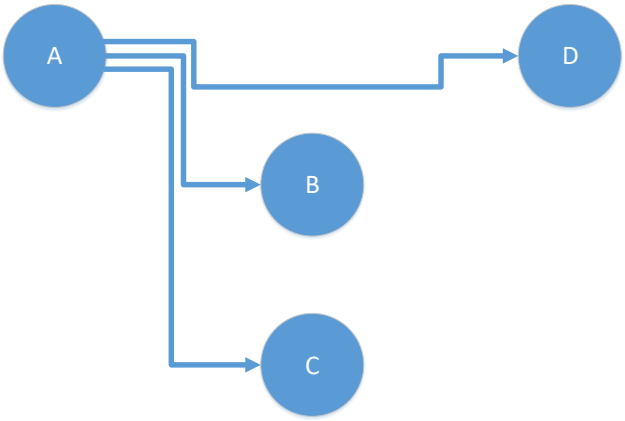


Рисунок 2.5 Правильне “рознесення” відрізків

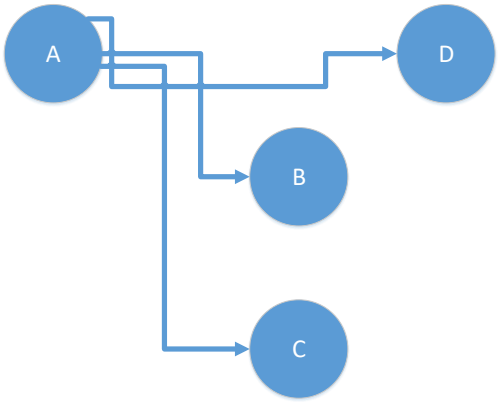


Рисунок 2.6. Неправильне “рознесення” відрізків

На цьому закінчується метод і отримане зображення задовольняє основні естетичні параметри візуалізації графа.

2.2.Етапи методу

1) Розташування і просторовий розподіл вузлів графа згідно їх ступеня віддаленості від вхідних портів (вузлів які мають тільки вихідні порти)
2) Побудова допоміжної мережі «шин», які проходять між вузлами і ребра якої слугуватимуть прототипом для створення відрізків з’єднань графа.
3) Побудова усіх з’єднань графа через мережу «шин» з алгоритму пошуку найкоротшого шляху з застосуванням вагових коефіцієнтів.
4) Розгортання шин у з’єднання, що не накладаються одне на одне.

2.3.Загальний алгоритм виконання

На першому кроці необхідно розташувати вузли графа у просторі. При цьому необхідно вершини будуть розташовуватися порівнево та без зворотних зв’язків (зв’язки не можуть виходити із наступного рівня з попередній). На даному кроці методу доцільно використати наступний алгоритм.

На перший рівень поміщуються вершини, які мають лише вихідні зв’язки. Після цього вершини із першого рівня необхідно видалити з початкового графа, а також видалити вихідні зв’язки, котрі виходили із видалених вершин. Після цього знову проводиться пошук вершин без вхідних зв’язків. Даний алгоритм повторюється, поки не будуть пройдені усі вершини.

На другому кроці побудуємо допоміжну мережу “шин”. Для створення цієї мережі “шин” спочатку проведемо вертикальні прямі, посередині між кожним вертикальним набором вершин. Далі, між кожною вертикальною парою вершин, проведемо горизонтальний відрізок до перетину із попередньо проведеними вертикальними лініями. Таким чином, відрізки допоміжної мережі “шин” будуть рівновіддаленими від вершин. В результаті цих дій отримаємо зображення, подібне до Рисунок 2.3. В точках перетину відрізків будуть розташовані вершини допоміжного графа, а самі відрізки будуть прототипами для зв’язків між вершинами графа. Також, підмітимо що для забезпечення ортогональності відрізків, з яких складається зв’язок, необхідно щоб відрізки допоміжної мережі “шин” перетиналися під “прямим кутом”, тобто вони повинні бути перпендикулярними. В загальному випадку, не обов’язково щоб відрізки допоміжної мережі “шин” були рівновіддаленими. Єдина вимога до цих відрізків – їх перпендикулярність.

На наступному кроці відбувається пошук найкоротших маршрутів між вершинами графа з допомогою попередньо створеної допоміжної мережі “шин”, а точніше з допомогою вершин допоміжного графа. Для обчислення найкоротшого шляху у графі використовується модифікований алгоритм Дейкстри. Будемо рухатися вниз та вправо по утвореній просторовій структурі вершин графа та виконуватимемо алгоритм Дейкстри почергово для усіх входів та виходів вершин.

Для усунення накладань відрізків на мережі “шин” необхідно розгорнути шини у з’єднання, що не накладаються одне на одне. Для цього використаємо наступний алгоритм:

1. Визначимо довжину та характер кожного маршруту від початкової вершини до кінцевих вершин.
2. Обчислимо довжину кожного відрізка від початкової точки до наступного “повороту” (зміни напрямку лінії). Запам’ятаємо довжини відрізків та напрямок поворотів для усіх складових зв’язку від початкової до кінцевої точки.
3. Розіб’ємо значення, отримані у кроці 2 на дві групи: відрізки, які “повернуть” ліворуч (масив “лівих”) та відрізки, які “повернуть” праворуч (масив “правих”).
4. Сортуємо масив “лівих” за зростанням довжини, масив “правих” за спадання довжини. Якщо для певних точок довжина є однаковою, то проводимо паралельні відрізки і потім визначимо як краще провести зв’язок.
5. Доповнюємо масив “лівих” елементами із масиву “правих”. В результаті отримано порядок у якому повинні бути розташовані відрізки.
6. Кроки 2-5 повторюються для усіх відрізків від початкової точки до кінцевих з урахуванням попередньо обрахованих значень.

Наведемо приклад для кращого розуміння приведеного алгоритму. Припустимо що в результаті попередніх кроків виконання було отримано розташування вершин та зв’язків у графі, як на рис. 2.7.

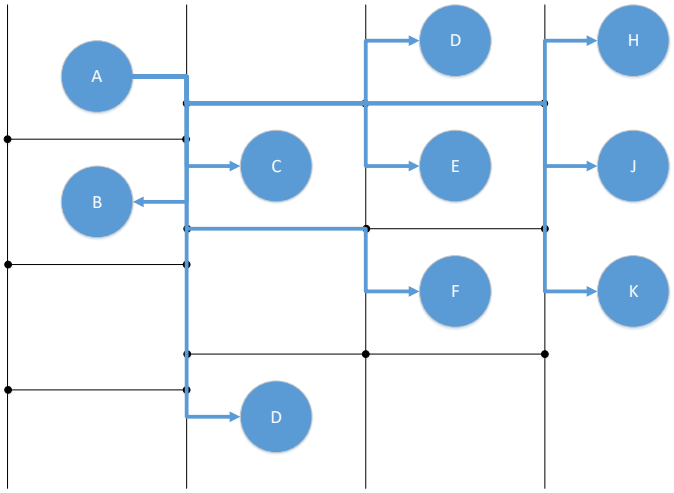


Рисунок 2.7. Початкове розташування вершин

Задача полягає в тому щоб рознести усі зв’язки від вершини А до інших вершин, уникаючи накладання зв’язків.

Із точки A до другої вертикальної “шини” усі зв’язки накладаються тому в даному випадку не можна нічого сказати про те як повинні слідувати відрізки. Тому далі переходимо до наступного відрізка. Далі необхідно визначити котрі відрізки “повертатимуть” праворуч, а які ліворуч. Так, праворуч повертає лише відрізок, який слідує до вершини B, усі ж інші повертають ліворуч.

Розташуємо кожен відрізок за спаданням довжини для відрізків “лівої” групи. Так, найдовшим відрізком є частина зв'язку від вершини A до D. Далі слідує відрізок від A до F, за ним від A до C с від початкової точки до наступного “повороту” (зміни напрямку лінії). Вершини D, E, H, J, K мають однакову довжину відрізка на даному етапі, тому проведемо 5 паралельних ліній із вершини A та на наступному кроці визначимо як краще провести зв’язки. В результаті першої ітерації отримали зображення, таке як на Рисунок *2.8*.

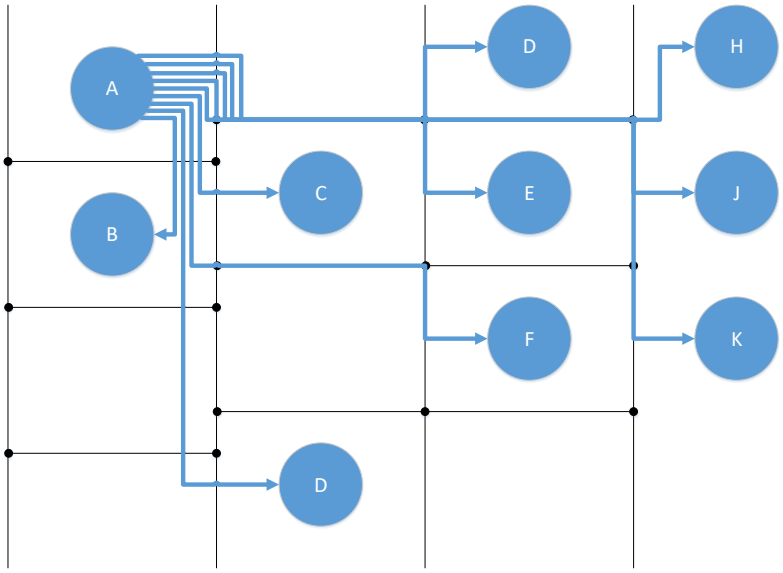
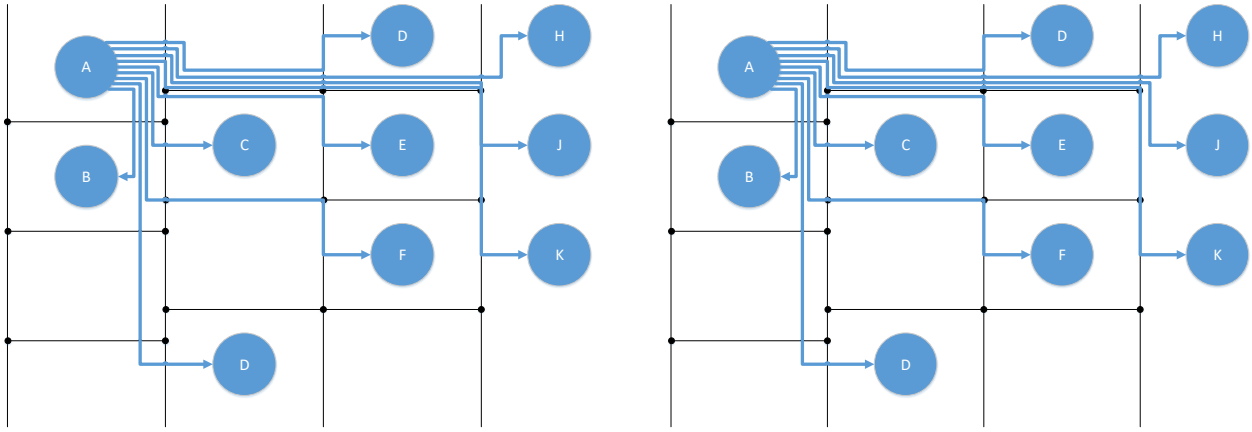


Рисунок 2.8. Граф після першої ітерації

На другій ітерації продовжуємо рухатися від вершини A до вершин D, E, H, J, K. Розбиваємо усі вершини на групи “лівих” та “правих”. Так, до “лівих” відносяться вершини D, H, до “правих” – E, J, K. Після сортування отриманих масивів отримаємо: “ліві” – D, H; “праві” – E, J, K. Виконуємо доповнення елементів масиву “лівих” елементами із масиву “правих”, отримаємо – D, H, E, J, K. В результаті другої ітерації отримали зображення, таке як на Рисунок *2.9*.

Залишилися лише вершини J та K. Виконуємо ті ж самі операції, що і на попередніх ітераціях.



а б

Рисунок 2.9. Граф після другої(а) та третьої(б) ітерації

В результаті було усунуто накладання відрізків на мережі “шин”. Також, візуалізовано граф запропонованим модифікованим методом знаходження найкоротших шляхів та прокладання з’єднань у орієнтованому ациклічному графі.

2.4.Висновки

У даному розділі було проведено опис запропонованого методу знаходження найкоротших шляхів та прокладання з’єднань у орієнтованому ациклічному графі. Аргументовано для чого потрібна кожна складова методу та що досягається в результаті її виконання. Також було сформульовано етапи із яких складається запропонований метод та створено алгоритм візуалізації з'єднань графа, який повинен виконувати поставлену мету та має задовольняти критеріям оптимальності. Також було наведено загальний алгоритм виконання методу детальним описом етапів методу та які дії необхідно виконувати на кожному етапі.

3. АПРОБАЦІЯ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1. Аналіз існуючих бібліотек для візуалізації графів

Розташування вершин і ребер графа не є простою задачею і може вимагати значної кількості витраченого часу в ручному режимі уже для графа із порядком 10 (порядок графа – кількість вершин).

Було проведено аналіз існуючих бібліотек (рішень) візуалізації графів. Зокрема, було зроблено аналіз наступних компонентів:

– GoXam – хороша бібліотека із великою кількістю вбудованих шаблонів для графів. Проте, слід виділити декілька мінусів: низька швидкодія при великій кількості вузлів графа, висока ціна за використання бібліотеки. Також, потрібно відмітити що в даному рішенні немає можливості для мінімізації накладань з’єднань графа (зображено на Рисунок 3.1);

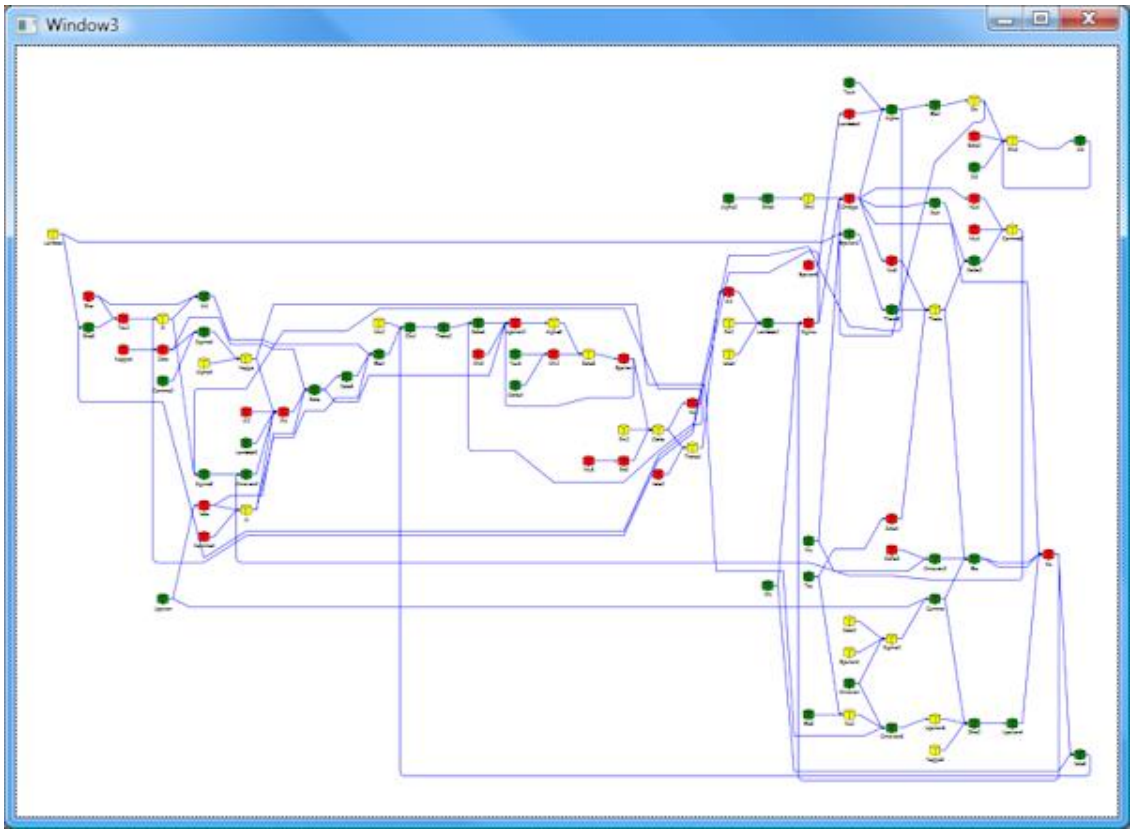


Рисунок 3.1. Візуалізація графа в GoXam

– Telerik – платне рішення для побудови графіків, графів та таблиць. Проте, в даній бібліотеці мала кількість шаблонів для побудови графів. Існуючі ж шаблони, не оптимізовані для роботи з великою кількістю даних і можуть нормально працювати лише на наборах до 100 вершин. Відмітимо, що дана бібліотека також не надає можливості для мінімізації накладань з'єднань графа (зображено на Рисунок 3.2).

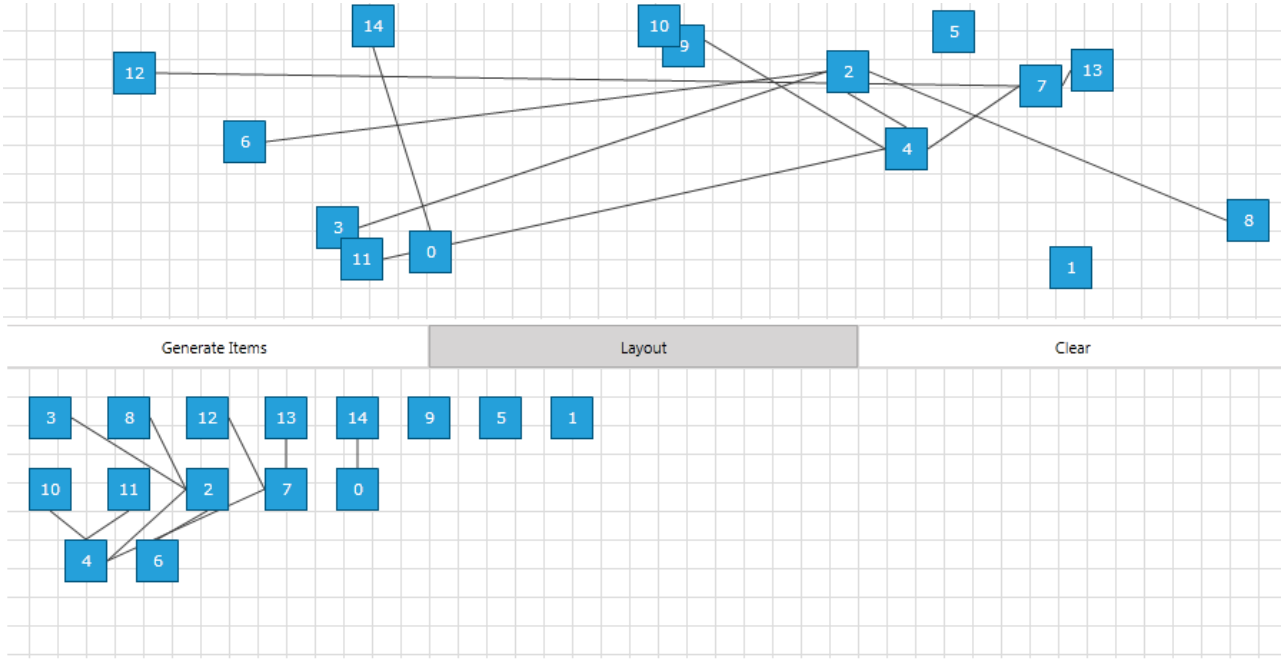


Рисунок 3.2. Візуалізація графа в Telerik

3.2.Опис технологій розробки

3.2.1. Трирівнева архітектура додатку

Клієнт-серверна архітектура – мережева архітектура, в якій мережеве навантаження розподілене між постачальниками послуг (серверами) та замовниками послуг (клієнт). Зазвичай клієнт та сервер розташовані на різних обчислювальних машинах і взаємодіють між собою через мережу за допомогою мережевих протоколів, але вони можуть бути розташовані також і на одній машині.

Логіка додатку розподілена між сервером і клієнтом. Сервер містить певну інформацію та може ділитися нею із клієнтами. В свою чергу клієнти не надають

доступ до своїх файлів/ресурсів, проте можуть отримати ресурси веб-сервера з допомогою веб-запиту. Зберігання даних відбувається, переважно, на сервері, обмін інформацією відбувається по мережі.

Далі опишемо основні переваги клієнт-серверної архітектура додатків:

– Швидкодія у розробці додатків та їх використанні. Так як зберігання даних відбувається на веб-сервері, то дані є наближеними до обчислювальних процесів. Велика частика розрахунків відбуваються на цьому ж сервері (в дворівневій архітектурі) чи на іншому сервері із високими обчислювальними можливостями. Це сприяє збільшенню швидкодії програмного забезпечення. Також дана архітектура дозволяє розробникам створювати прикладні системи для клієнт-серверної архітектури в дуже короткі терміни.

– Висока безпека. Технологія клієнт-сервер дає велику безпеку для користувача, процеси виконуються більш стабільно, злагоджено, з підвищеною конфіденційністю, дані швидко обробляються і надійно зберігаються [4].

– Клієнтська частина програми (та, що знаходиться у користувача), передає по мережі запит на обробку даних на інший комп'ютер, на якому сервер бази даних їх читає, виконує, і повертає по мережі відповідь клієнту. Причому по мережі передається лише корисна інформація.

– Мультиплатформенність. Клієнти не залежать від конкретної операційної системи користувача.

– Відсутність дублювання коду додатку клієнтами. Основна логіка бізнес-процесів та обробки даних знаходиться на сервері. Клієнт, в свою чергу, головним чином виступає для підвищення інтерактивності додатків, наприклад для перевірки коректності даних, що вводяться без додаткового звернення до сервера, і для створення зручного для користувача інтерфейсу.

– Так як всі обчислення виконуються на сервері, то вимоги до комп'ютерів, на яких встановлено клієнт, знижуються. За рахунок цього також знижується собівартість усієї системи.

– Всі дані зберігаються на сервері, який, як правило, захищений набагато краще за більшість клієнтів. На сервері простіше організувати контроль доступу, щоб надавати до дані тільки клієнтам з відповідними правами.

Зазвичай використовується трьохрівнева архітектура «клієнт-сервер», котра включає в себе клієнт, сервер додатків та сервер бази даних. Схематично, схема такої архітектури представлена на рисунку Рисунок 3.3.

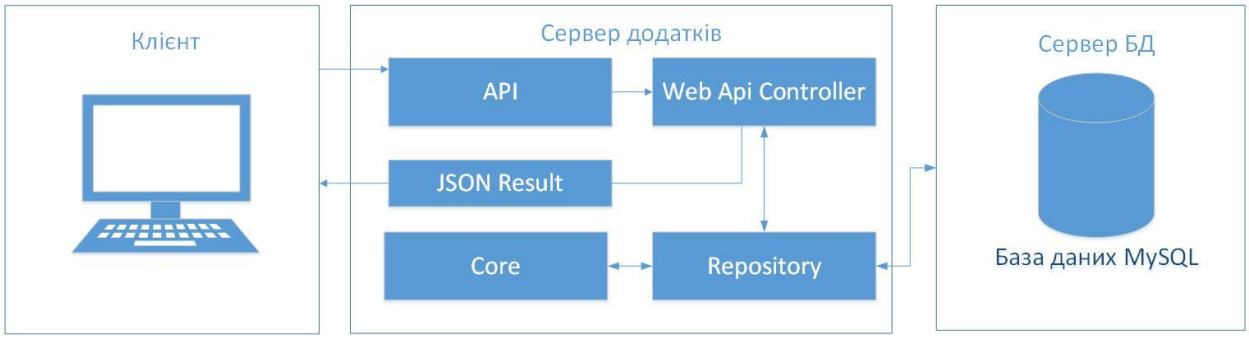


Рисунок 3.3. Трьохрівнева архітектура

Далі наведемо короткий опис кожного компонента архітектури:

1. Клієнт представляє собою застосунок для кінцевого користувача. Клієнтська частина є реалізацією так званих тонких клієнтів — логіка застосунку зосереджується на сервері, а функція браузера полягає переважно у відображенні інформації, завантаженої мережею з сервера, і передачі назад даних користувача. Сервер може обслуговувати кількох клієнтів одночасно. У цьому випадку говорять про багатокористувацький режим. Тільки не варто розуміти слово "одночасно" в буквальному сенсі. Запити виконуються сервером послідовно.

2. Сервер додатків діє як набір компонентів, доступних розробнику через API, визначений самою платформою. На цьому рівні зосереджена більша частина бізнес-логіки. Сервер отримує запит від клієнта і виконує обчислення, після цього формує веб-сторінку і відправляє її клієнтові мережею з використанням протоколу HTTP. Веб-сервер в свою чергу звертається до бази даних, обробляє дані та відсилає їх клієнту.

3. Сервер БД забезпечує зберігання даних. та є джерелом усієї корисної інформації для системи. Взаємодія з БД відбувається через сервер додатків, за допомогою SQL-запитів.

Така архітектура не дозволяє клієнтській частині звертатися безпосередньо до бази даних, що забезпечує високий рівень безпеки та надійність усієї системи.

Як будь-яка працююча система, трьохрівнева архітектура має свої переваги та недоліки. Так, серед головних переваг трьохрівневої моделі виділяють:

- Масштабованість. Сервер бази даних міститься на віддаленому сервері, до якого по мережі підключені один або кілька серверів додатків, до яких, в свою чергу, по мережі підключаються клієнти. Тобто при високих навантаженнях можна із легкістю збільшити кількість серверів додатків. Розрізняють горизонтальне та вертикальне масштабування.
- Конфігурованість — ізольованість рівнів один від одного дозволяє швидко і простими засобами настроювати систему при виникненні збоїв на одному з рівнів.
- Високий рівень безпеки.
- Висока надійність.
- Єдиний інтерфейс. Єдиний інтерфейс між клієнтами та серверами спрощує і розділяє архітектуру, дозволяючи кожному компоненту розвиватися самостійно. Також потрібно відмітити що клієнтом може виступати веб-браузер, десктопний чи мобільний додаток, це досягається саме завдяки єдиному інтерфейсу.

До недоліків зазвичай відносять:

- Більш висока складність створення додатків;
- Складніше в розгортанні і адмініструванні;
- Високі вимоги до продуктивності серверів додатків і сервера бази даних, а, значить, і висока вартість серверного обладнання;
- Високі вимоги до швидкості каналу (мережі) між сервером бази даних і серверами додатків.

3.2.2. Рівень представлення

Для реалізації рівня представлення в даній системі були використані JavaScript фреймворк Angular та HTML, CSS фреймворк Bootstrap.

3.2.2.1. TypeScript

TypeScript - мова програмування, представлений Microsoft в 2012 році і позиціонується як засіб розробки веб-додатків, що розширює можливості JavaScript.

Специфікації мови відкриті і опубліковані в рамках угоди Open Web Foundation Specification Agreement. TypeScript є сумісним з JavaScript і компілюється в останній. Фактично, після компіляції програму на TypeScript можна виконувати в будь-якому сучасному браузері або використовувати спільно з серверної платформою Node.js. Код експериментального компілятора, який транслює TypeScript в JavaScript, поширюється під ліцензією Apache. Його розробка ведеться в публічному репозиторії через сервіс GitHub.

TypeScript відрізняється від JavaScript можливістю явного статичного призначення типів, підтримкою використання повноцінних класів (як в традиційних об'єктно-орієнтованих мовах), а також підтримкою підключення модулів, що покликане підвищити швидкість розробки, полегшити читаність, рефакторинг і повторне використання коду, допомогти здійснювати пошук помилок на етапі розробки і компіляції, і, можливо, прискорити виконання програм.

Планується, що в силу повної сумісності адаптація існуючих додатків на нову мову програмування може відбуватися поетапно, шляхом поступового визначення типів.

3.2.2.2. Angular

Angular представляє собою фреймворк від компанії Google для створення клієнтських додатків. Перш за все він націлений на розробку SPA-рішень, тобто односторінкових додатків. В цьому плані Angular є спадкоємцем іншого

фреймворка AngularJS. У той же час Angular це не нова версія AngularJS, а принципово новий фреймворк.

Angular це платформа та фреймворк для створення клієнтських додатків базуючись на HTML та TypeScript. Angular було реалізовано мовою програмування TypeScript. Фреймворк реалізує основні та додаткові функції як набір бібліотек TypeScript, що імпортуються в додатки.

Angular надає таку функціональність, як двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому, шаблони, маршрутизація і так далі.

Однією з ключових особливостей Angular є те, що він використовує в якості мови програмування TypeScript. Angular не обмежений мовою TypeScript. При бажанні можна писати програми на Angular за допомогою таких мов, як Dart або JavaScript. Однак TypeScript все таки є основною мовою для Angular.

Angular дозволяє з "коробки", тобто без значної конфігурації, створювати великі і складні за частиною бізнес-логіки додатка. Angular було повним переосмисленням AngularJS, напевно, це було найболючіше, але воно того варте, сам фреймворк став куди чистішим і гнучкішим, більш enterprise-подібним і з цієї точки зору має високу масштабованість.

Основними блоками додатку, створеному на Angular є NgModules, які надають контекст компіляції для компонент. NgModules збирають відповідний код у свого роду набори, в залежності від функціональності. Angular додаток визначається набором NgModules. Додаток завжди має як мінімум кореневий модуль, який дозволяє завантажуватися додатку, і зазвичай має багато інших функціональних модулів.

Компоненти визначають представлення, які є наборами елементів для відображення, що можна вибирати серед і змінювати відповідно до вашої логіки додатку або даних.

Компоненти користуються сервіси (service), які забезпечують функції, не пов'язані безпосередньо з представленнями. Сервіси можуть бути введені в

компоненти як залежності, що робить код модульним, ефективним і надає можливість повторного використання.

Компоненти та сервіси є просто класами, що містять декоратори, які відмічають їх тип, і надають метадані, за допомогою яких Angular визначає, як їх треба використовувати.

Метадані класу компонент пов'язують його з шаблоном, який визначає представлення. Шаблон об'єднує звичайний HTML з директивами Angular та розмітку, що дозволяє фреймворку змінювати HTML перед відображенням. Метадані класу сервісу надають інформацію Angular, за допомогою якої можна зробити її доступною для компонент за допомогою ін'єкцій залежностей.

Компоненти програми зазвичай визначають багато представлень, розташованих ієрархічно. Angular забезпечує службу маршрутизатора, щоб допомогти вам визначити шляхи навігації серед представлень. Клас роутер забезпечує складні навігаційні можливості в браузері.

Все, що потрібно знати про Angular для розробки продуктивних і швидких додатків будь-якого рівня складності, описано нижче в наступних концепціях:

Form Builder – для розробки справді складних форм, потрібно знати реактивні форми, точніше принципово забути про декларативні форми.

Change Detection – так як Angular за замовчуванням використовує двостороннє зв'язування моделі даних, то при роботі з великим об'ємом таких даних програми будуть працювати повільніше, тому в деяких випадках варто подбати про правильну стратегію виявлення змін, а саме: ChangeDetection.OnPush.

Templating – синтаксис шаблонів з точки зору абстракції змінився не дуже сильно в порівнянні з AngularJS. Мається на увазі що можна так саме написати умови, цикли, зв'язати модель даних та інше. Все, що вам слід добре зрозуміти і розібратися в Angular шаблонах - що таке структурні і декларативні директиви, а також що таке Input-параметри і Output-події.

Routing – напевно, це одне з основних явищ у розробці веб-додатків. Тут просто важливо розуміти, що маршрутизація так само, як і компоненти, має свій

життєвий цикл, розуміючи це, можна писати справді хороші додатки. Ще варто відзначити: якщо на якийсь із маршрутів повішано модуль, а не компонент, який відповідає за відображення сторінки по цьому маршруту, то модуль буде довантажувати на сторінці на вимогу.

Annotations – декоратори, які використовуються в достатку при написанні додатків на Angular, не є якоюсь жорсткою магією TypeScript. Декоратори є специфікацією EcmaScript і коли браузери почнуть підтримувати їх, вони будуть нативно виконуватися в browser runtime. Насправді, декоратори вельми корисні і забезпечують досить високу читаємість написаного коду. Один із прикладів - це валідація моделей даних з використанням декораторів або десеріалізацію/сериализация даних.

Observables – насправді, тут варто відзначити тільки те, що незабаром Observables будуть специфікацією EcmaScript і все це буде нативно підтримуватися в браузерах. З точки зору теорії, якщо розкрити поняття Observer (спостерігач) - це поведінковий шаблон проектування. У класі створюється механізм, який дозволяє екземпляру об'єкта цього класу отримувати оповіщення від інших об'єктів про зміну їх стану, тим самим спостерігаючи за ними.

Shadow DOM – це засіб для створення окремого DOM-дерева всередині елемента, яке не видно зовні без застосування спеціальних методів. Грубо кажучи, це зручний спосіб створення ізольованих веб-компонентів, котрі можна використовувати повторно. Чисто технічно, якби браузери вже сьогодні підтримували багато концепцій, які використовує Angular, нам не потрібні були б транспайлери і інші системи збирання, все, що ми писали на Angular, працювало б уже нативно.

3.2.2.3. Bootstrap

Bootstrap - це інтуїтивно зрозумілий та потужний фронтенд фреймворк, що орієнтований на мобільність і використовується для швидкого і просте створення веб-сторінок. Він побудований на основі HTML, CSS та JavaScript.

Фреймворк був розроблений Марком Отто та Джейкобом Торнтоном з компанії Twitter. В серпні 2011 року він був випущений як продукт з відкритим вихідним кодом на GitHub.

В фреймворку Bootstrap є ряд переваг, які допомагають в розробці клієнтської частини додатку, а саме:

- Зосередження на мобільності - Bootstrap містить структуру стилів, які допомагають мобільності і знаходять в одному файлі замість поділу на окремі
- Підтримка браузерів - підтримується всіма популярними браузерами
- Легко почати роботу. Маючи знання HTML та CSS можна швидко почати роботу з Bootstrap. Також можна знайти вичерпну документацію на офіційному сайті Bootstrap
- Адаптивний дизайн - адаптивний CSS працює на настільних комп'ютерах, планшетах та мобільних пристроях
- Забезпечує чисте та однорідне рішення для створення інтерфейсу розробниками
- Він містить гарні та функціональні вбудовані компоненти, які легко налаштовувати
- Надає можливість веб-кастомізації
- Має відкритий сирцевий код

В пакет фреймворку входять наступні компоненти:

- Каркас - Bootstrap забезпечує базову структуру з системою розмітки, стилями та фоном.
- CSS - Bootstrap поставляється з функцією глобальних налаштувань CSS, основних HTML-елементів, стилізованих та розширених CSS класами, а також розширеної системи розмітки.
- Компоненти – фреймворк містить більше десятка компонент для багаторазового використання, створені для забезпечення ілюстрацій, випадаючих меню, навігації, сповіщень, спливаючих вікон та багато іншого.

– Плагіни JavaScript - Bootstrap містить більше десятка користувальницьких плагінів jQuery. Їх можна легко включити усі або один за одним.
– Кастомізація – існує можливість налаштувати компоненти Bootstrap, LESS змінні та jQuery плагіни, щоб отримати власну версію.

Як уже було сказано, зв’язок між прикладним рівнем та рівнем представлення було реалізовано, як RESTful API.

3.2.3. Прикладний рівень

3.2.3.1. Мова програмування C#

C# — мультипарадигмана мова програмування з безпечною системою типізації для платформи .NET, котра підтримує об'єктно–орієнтовану, функціональну, імперативну, декларативну, подійно–орієнтовану парадигми.

C# є однієї із мов програмування, котра спроектована для CLR (Common language runtime). Тобто, кожна мовна конструкція транслюється у конструкції CLR. Дана особливість накладає обмеження на внесення змін у мову C#.

Оскільки кожна конкретна особливість мови має бути трансльована у відповідні конструкції CLR, то C# тому вона залежить від можливостей CLR (перш за все, це стосується фсистеми типів C#) [1].

C# багато перейняв у Java і C++. C# підтримує наслідування, поліморфізм, перевантаження операторів, строгу статичну типізацію, атрибути, події, виключення тощо. Об'єктно–орієнтований підхід дозволяє вирішити завдання з побудови великих, але у той же час гнучких, масштабованих і розширюваних додатків. C# продовжує активно розвиватися, і з кожною новою версією з'являється все більше цікавих функціональних можливостей, як, наприклад, асинхронні методи, лямбда–функції, кортежі, інтерполяція строк, динамічне зв’язування тощо.

Мова програмування C# увібрала у себе багато позитивних рис своїх попередників, були вилучені моделі, котрі показали свою нестабільність чи

зарекомендували себе як проблематичні в інших мовах програмування(наприклад множинне спадкування класів у C++) [2].

Також необхідно відзначити що синтаксис мови C# постійно розширюється (особливо в останніх версіях мови). Наприклад, були додані наступні можливості: null–умовні операції, ініціалізатори властивостей і індексів, інтерполяція строк, кортежі, локальні функції, зіставлення шаблонів тощо. Внаслідок цього код виглядає більш простим та лаконічним.

Мова C# актуальна в першу чергу тому, що дозволяє більш раціонально створювати популярні на сьогодні інтернет–додатки. Мова C# тісно інтегрована з різними веб–технологіями. Мова C# містить в собі переваги мови Java і C ++, що й обумовлює популярність даного мови серед розробників. При цьому в об'єднаному мовою виключені деякі спірні директиви, макроси, скасовані глобальні змінні.

Можна підсумувати, що C# дозволяє написати код, який одночасно задовольняє параметри «простоти розробки», «краси коду» і продуктивності.

3.2.3.2. Фреймворк ASP.NET Web API

ASP.NET — серверна технологія, із відкритим вихідним кодом, для створення динамічних веб–сайтів, веб–додатків і веб–сервісів. Вона є складовою частиною платформи Microsoft.NET.

Технологія ASP.NET побудована на основі CLR, а тому розробники можуть писати код, практично, на будь–якій мові програмування, що входять у комплект .NET Framework. При першому зверненні код компілюється і поміщається в спеціальний кеш, і згодом тільки виконується, не вимагаючи витрат часу на парсинг чи оптимізацію. Завдяки цьому ASP.NET має перевагу у швидкості в порівнянні зі скріптовими технологіями.

ASP.NET дає можливість просто створити REST API. REST – це загальні принципи організації взаємодії додатка/сайту з сервером за допомогою протоколу HTTP. Особливість REST в тому, що сервер не запам'ятовує стан користувача між запитами – в кожному запиті передається інформація, що

ідентифікує користувача (наприклад, token, отриманий через OAuth–авторизацію) і всі параметри, необхідні для виконання операції. Уся взаємодія з сервером зводиться до 4 операцій: отримання даних з сервера(GET), додавання нових даних на сервер (POST), модифікація існуючих даних на сервері (PUT) та видалення даних на сервері (DELETE).

Сьогодні найбільш актуальним способом створити REST сервіс у стеку технологій .NET є ASP.NET Web API.

ASP.NET Web API містить HTTP і робить розробку RESTful сервісів надзвичайно простою. Також дана технологія має відкритий код, а це в свою чергу сприяє кращому зворотному зв'язку із користувачами. Користувачі можуть вносити зміни у проект, виправляти помилки, висловлювати побажання щодо покращення фреймворку [3].

Також перевагою даної технології є те що вона побудована за тими ж принципами, що й ASP.NET MVC, що дозволяє легко перейти з однієї технології на іншу.

ASP.NET Web API має вбудовану підтримку для наступних форматів: JSON, BSON, XML. При взаємодії з клієнтом між ним і сервером відбуваються два процеси: узгодження вмісту і серіалізація вмісту). На першому етапі сервер дізнається, в якому форматі клієнту краще відправляти відповідь. На другому етапі відбувається серіалізація даних в цей формат. Тобто це означає, що клієнти можуть використовувати різні формати передачі даних [4].

3.2.3.3. Платформа Microsoft Azure

Microsoft Azure – хмарна платформа та інфраструктура, яку розробники та IT–фахівці використовують для розробки та розгортання додатків, а також управління ними через нашу всесвітню мережу центрів обробки даних [5]. Завдяки Azure можна створювати та розгортати рішення в будь–якому місці, використовуючи необхідні інструменти, програми та платформи.

Служба додатків Azure – це нова та унікальна хмарна служба, яка дозволяє розробникам створювати веб–додатки та мобільні додатки для будь–якої

платформи та будь–якого пристрою. Для служби додатків Azure можна налаштувати робочий процес безперервного розгортання. Служба додатків Azure інтегрується з BitBucket, GitHub і Visual Studio Team Services (VSTS), що забезпечує безперервне розгортання, коли Azure отримує останні оновлення з проекту, опубліковані в одній з цих служб [6]. Безперервне розгортання дуже зручно для проектів, які часто оновлюються декількома учасниками.

3.2.3.4. Менеджер пакетів NuGet

NuGet – це система керування пакунками, що дозволяє розробнику легко керувати бібліотеками в проектах будь–якого типу. Основним завданням NuGet є спрощення процесу встановлення сторонніх бібліотек. Він здійснює пошук бібліотеки, завантаження необхідного для установки пакета, перевірку цілісності пакета, розпакування файлів бібліотеки в потрібне місце, додавання посилань на її збірки в проект та модифікацію файлів конфігурації (web.config, app.config) при необхідності [7].

3.2.3.5. Специфікація OWIN та Katana

OWIN – специфікація, яка усуває сильну зв'язаність веб–додатки з реалізацією сервера. Вона дозволяє запускати додатки на будь–якій платформі, що підтримує OWIN, без змін. Насправді, специфікація дуже проста – це просто «словник» з параметрів і їх значень. Базові параметри визначені в специфікації [3].

Раніше, до створення специфікації OWIN, технологія ASP.NET була сильно зв’язана із веб–сервером IIS, тому було дуже складно розвернути веб–додаток на іншому сервері.

Розробники веб–серверів можуть бути впевнені в тому, що, якщо вони реалізують OWIN правильно, то додатки ASP.NET будуть коректно працювати на їхньому сервері [8].

Стандартною реалізацією OWIN є Katana. Вона дозволяє розміщувати OWIN–збірки в IIS і у вигляді самостійного додатка(self–host).

3.2.3.6. Веб–сервіс для хостингу Atlassin Bitbucket

Git — розподілена система керування версіями файлів та спільної роботи. Git надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні та злитті гілок. Також Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями.

Система має ряд користувацьких інтерфейсів: наприклад, gitk та git–gui розповсюджуються з самим Git [9].

Також існують сервіси для хостингу проектів на базі Git. Найпопулярнішими серед ним є Github та Bitbucket.

Bitbucket — веб–сервіс для хостингу проектів на базі систем керування версіями: Mercurial та Git.

Bitbucket дозволяє створювати необмежену кількість безкоштовних приватних репозиторіїв (до 5 користувачів), на відміну від GitHub, де ця можливість є платною [10]. Також корисною можливість системи є функція «Side–by–side diff», котра наглядно показує внесені зміни у коді.

3.2.3.7. Засіб тестування NUnit

Nunit — фреймворк з відкритим вихідним кодом для модульного тестування усіх мов програмування .Net. Дана технологія дозволяє перевірити коректність окремих модулів(інтерфейсів, класів) вихідного коду програми. Ідея полягає в тому щоб написати тести для кожного нетривіального методу.

Даний підхід дозволяє швидко перевірити чи чергова зміну коду не призвела до появи помилок у вже протестованих місцях програми, а також полегшує виявлення та усунення помилок.

Переваги фреймворку NUnit:

- Написані тести можуть бути запущені як із консолі, так і з IDE;
- Тести запускаються паралельно, що пришвидшує їх виконання;
- Хороша підтримка ASP.NET;
- Якщо додаток зберігає налаштування в файлах конфігурації, то NUnit надає можливість створити окремий файл конфігурації для тестування.

3.2.4. Рівень даних

3.2.4.1. Мова запитів SQL

SQL (Structured Query Language – Структурована мова запитів) – мова управління базами даних для реляційних баз даних. Сам по собі SQL не є повною мовою програмування, але його стандарт дозволяє створювати для нього процедурні розширення, які розширюють його функціональність до повноцінної мови програмування [11].

Не зважаючи на існування стандартів, більшість поширених реалізацій SQL відрізняються так сильно, що код рідко може бути перенесений з однієї СУБД в іншу без внесення істотних змін. Це пояснюється великим обсягом і складністю стандарту, а також браком в ньому специфікацій в деяких важливих областях реалізації.

SQL створювався як простий стандартизований спосіб отримання та управління даними, що містяться в реляційній базі даних. Пізніше він став складнішим, ніж замислювався, і перетворився в інструмент розробника, а не кінцевого користувача.

SQL складається з чотирьох окремих частин:

мова визначення даних (DDL) використовується для визначення структур даних, що зберігаються в базі даних. Оператори DDL можна створювати, редагувати та видаляти окремі об'єкти в БД. Допустимі типи об'єктів залежать від використовуваної СУБД і зазвичай включають бази даних, користувачів, таблиці та ряд дрібніших допоміжних об'єктів, наприклад, ролі та індекси.

– мова маніпуляції даними (DML) використовується для отримання і зміни даних в БД. Оператори DML дозволяють отримувати, вставляти, змінювати та видаляти дані в таблицях. Іноді оператори select вилучення даних не розглядаються як частина DML, оскільки вони не змінюють стан даних. Всі оператори DML мають декларативний характер.
– мова визначення доступу до даних (DCL) використовується для контролю доступу до даних в БД. Оператори DCL застосовуються до привілеїв і

дозволяють видавати та відбирати права на застосування певних операторів DDL і DML до певних об'єктів БД.

– мова управління транзакціями (TCL) використовується для контролю обробки транзакцій в БД. Зазвичай оператори TCL включають commit для підтвердження змін, зроблених в ході транзакції, rollback для їх скасування та savepoint для розбиття транзакції на кілька менших частин [12].

3.2.4.2. СУБД MySQL

MySQL – це вільна система керування базами даних, що була створена як альтернатива комерційним системам. Вона використовує мову SQL для комунікації з іншими програмами. Проте, MySQL має свої власні розширені функції SQL для того щоб забезпечити користувачам додатковий функціонал. MySQL однією із найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб–сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування [13].

MySQL — компактний багатопотоковий сервер баз даних. Характеризується високою швидкістю, стійкістю і простотою використання.

MySQL вважається гарним рішенням для малих і середніх застосувань. Сирцеві коди сервера компілюються на багатьох платформах. Найповніше можливості сервера виявляються в UNIX–системах, де є підтримка багатопоточності, що підвищує продуктивність системи в цілому.

3.3.Опис тестових випадків

Необхідно провести експерименти для того щоб перевірити ефективність запропонованого методу. Зокрема, потрібно щоб виконувалися поставлені задачі та результати задовольняли висунутим критеріям оптимальності. З цією метою необхідно провести експерименти із використанням запропонованого методу та без нього. У другому випадку будемо використовувати існуючі аналоги (GoXam і Telerik).

Визначимо наступний набір тестових випадків:

– Набір №1: малий набір даних (50 вузлів та 100 з’єднань);

– Набір №2: середній набір даних (200 вузлів та 400 з’єднань);
– Набір №3: великий набір даних (500 вузлів та 1000 з’єднань).

У кожному із запропонований випадків будемо вимірювати час, необхідний для побудови графу та можливість ефективної взаємодії користувача із отриманим зображенням (мається на увазі чи відбуваються операції “збільшення/зменшення”, “перетягування” зображення за прийнятний час).

Наступним параметром є ортогональність відрізків, з яких складається з’єднання графа. Тобто, мається на увазі що з’єднання розбиті на ортогональні відрізки, які повинні розташовуватися таким чином щоб уникати перетину з вузлами графа при його візуальному відображенні.

3.3.1. Граф без використання методу

В першому випадку перевіримо швидкість роботи бібліотеки GoXam. Для цього будемо використовувати Засікатимемо час від моменту коли дані передаються у елемент Virtualized Tree Layout та поки усе зображення не буде побудовано та доступне для наступних дій. Для кожного набору даних виконаємо 8 тестів щоб уникнути похибки вимірювань. Також, обрахуємо середнє значення, яким надалі будемо оперувати. Результати представлені у Таблиця 3.1.

Таблиця 3.1.Час візуалізації графа бібліотекою GoXam

Набір / №тесту	1	2	3	4	5	6	7	8	Середнє
1	303	255	216	138	313	236	194	165	227.50
2	1370	1342	1222	1373	1268	1473	1405	1342	1367.50
3	8168	8569	8087	8043	8301	8176	8238	8297	8214.10

Операції “збільшення/зменшення”, “перетягування” зображення за прийнятний час для малого та середнього наборів даних. Відрізки із яких складаються з’єднання графа не є ортогональними.

Повторимо ті ж самі вимірювання для бібліотеки Telerik. Результати вимірювань представлені у Таблиця 3.2.

Таблиця 3.2. Час візуалізації графа бібліотекою Telerik

Набір / №тесту	1	2	3	4	5	6	7	8	Середнє
1	114	159	174	157	127	184	169	178	157.75
2	238	269	276	253	242	239	245	241	250.38
3	1095	2243	1730	1732	1683	2131	1665	1677	1744.50

Операції “збільшення/зменшення”, “перетягування” зображення за прийнятний час для всіх наборів даних. Відрізки із яких складаються з’єднання графа

3.3.2. Граф з використанням методу

Тепер зробимо ті самі виміри для запропонованого методу. Див. додаток А, як вигладає зображення отриманого графу. Результати вимірювання представлені у Таблиця 3.3.

Таблиця 3.3. Час візуалізації запропонованим методом

Набір / №тесту	1	2	3	4	5	6	7	8	Середнє
1	67	67	61	66	72	64	66	64	65.88
2	142	167	165	182	176	166	187	160	168.13
3	966	760	826	776	829	783	805	843	823.50

Операції “збільшення/зменшення”, “перетягування” зображення за прийнятний час для всіх наборів даних. Відрізки із яких складаються з’єднання графа є ортогональними.

3.4.Порівняння результатів

Із отриманих результатів можна зробити висновок що з допоиогою запропонованого методу можна візуалізувати граф удвічі швидше ніж за допомогою бібліотеки Telerik, та у 10 разів швидше ніж за допомогою бібліотеки GoXam. Також, лише за допомогою реалізованої бібліотеки можна досягнути ортогональності відрізків із яких складається з’єднання між вершинами графа.

3.5. Висновки

В даному розділі було здійснено аналіз існуючих бібліотек для візуалізації графа та наведено їх недоліки. Було описано вимоги до власної бібліотеки, яка реалізовує запропонований метод. На основі обраної архітектури та технологій для розробки було розроблено бібліотеку, яка реалізує запропонований метод знаходження найкоротших шляхів та прокладання з’єднань у орієнтованому ациклічному графі. Також було проведено експерименти для доведення ефективності запропонованого методу. В результаті було отримано, що реалізована бібліотека будує зображення графа удвічі швидше за бібліотеку Telerik, та у десять разів швидше у порівнянні із бібліотекою GoXam. Відмітимо, що лише реалізована бібліотека будує з’єднання у вигляді ортогональності ортогональних відрізків.

4. МАРКЕТИНГОВИЙ АНАЛІЗ СТАРТАП-ПРОЕКТУ

Розроблення та виведення стартап-проекту на ринок передбачає здійснення низки кроків, в межах яких визначають ринкові перспективи проекту, графік та принципи організації виробництва, фінансовий аналіз та аналіз ризиків і заходи з просування пропозиції для інвесторів. Узагальнено етапи розроблення стартап-проекту можна подати таким чином:

1. Маркетинговий аналіз стартап-проекту

В межах цього етапу:

– розробляється опис самої ідеї проекту та визначаються загальні напрями використання потенційного товару чи послуги, а також їх відмінність від конкурентів;
– аналізуються ринкові можливості щодо його реалізації;
– на базі аналізу ринкового середовища розробляється стратегія ринкового впровадження потенційного товару в межах проекту.

2. Організація стартап-проекту

В межах цього етапу:

– складається календарний план-графік реалізації стартап-проекту;
– розраховується потреба в основних засобах та нематеріальних активах;
– визначається плановий обсяг виробництва потенційного товару, на основі чого формулюється потреба у матеріальних ресурсах та персоналі;
– розраховуються загальні початкові витрати на запуск проекту та планові загальногосподарські витрати, необхідні для реалізації проекту.

3. Фінансово-економічний аналіз та оцінка ризиків проекту

В межах цього етапу:

– визначається обсяг інвестиційних витрат;
– розраховуються основні фінансово-економічні показники проекту (обсяг виробництва продукції, собівартість виробництва, ціна реалізації, податкове навантаження та чистий прибуток) та визначаються показники інвестиційної привабливості проекту (запас

фінансової міцності, рентабельність продажів та інвестицій, період окупності проекту);

– визначається рівень ризикованості проекту, визначаються основні ризики проекту та шляхи їх запобігання (реагування на ризики).

4. Заходи з комерціалізації проекту

Цей етап спрямовано на пошук інвесторів та просування інвестиційної пропозиції (оферти). Він передбачає:

– визначення цільової групи інвесторів та опису їх ділових інтересів;
– складання інвест-пропозиції (оферти): стислої характеристики проекту для попереднього ознайомлення інвестора із проектом;
– планування заходів з просування оферти: визначення комунікаційних каналів та площадок та планування системи заходів з просування в межах обраних каналів;
– планування ресурсів для реалізації заходів з просування оферти.

Означені етапи, реалізовані послідовно та вчасно – створюють передумови для успішного ринкового старту. Проте фахівці зі створення та розвитку стартап-проектів окремо відзначають, що відсутність маркетингових знань та умінь, що уможливлюють розробку ринково затребуваного проекту із вихідної ідеї, є основною причиною високого рівня банкрутств стартап-компаній, і ця проблема може бути вирішена за рахунок навчання винахідників. Відповідно, основним призначенням даних Методичних рекомендацій є надання студентам знань щодо суті, основних принципів розроблення стратегії ринкового впровадження та маркетингового управління інноваційними стартап-проектами у промислових галузях економіки, використання ефективних маркетингових інструментів просування високотехнологічних продуктів виробництва та послуг.

4.1.Опис ідеї проекту

Розглянемо зміст ідеї, можливі напрямки застосування та основні вигоди, які може отримувати користувач від проекту.

Таблиця 4.1. Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Метод знаходження найкоротших шляхів та прокладання з'єднань між вузлами направленого графа	Бібліотека візуалізації графів	Можливість візуалізувати графи простим способом. Необхідно просто передати викликати функцію із бібліотеки та передати туди масив точок. Дане рішення також буде дуже гнучким.
	Частина платформи для відображення структур даних	Програмне забезпечення також можна використовувати як сервіс. Даний сервіс буде корисним для людей яким не цікава технологічна реалізація бібліотека, а котрі лише хочуть мати можливість отримати певне зображення структур даних при мінімальній кількості налаштувань.

Тепер визначимо які техніко-економічні переваги має даний додаток над іншими існуючими рішеннями. Проведемо порівняння з найближчими аналогами, а саме GoXam, Telerik.

Таблиця 4.2. Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№	Техніко-економічні характеристики ідеї	Продукція конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	GoXam	Telerik			
1	Ліцензія	Безкоштовна	Платна	Платна	Потрібно знаходити додаткові джерела фінансування		Дозволяє залучити більше нових клієнтів

Продовження таблиці 4.2

2	Кросплатформеність	Є	ОС Windows	ОС Windows			Дозволяє запускати додаток на будь-якій платформі з браузером
3	Open-source проект	Так	Ні	Ні			+
4	Можливість побудови з’єднань у вигляді ортогональних відрізків	Є	Нема	Нема			+
5	Безліч різних форм зображення графів	Нема	Є	Є	+		

Визначений перелік слабких, сильних та нейтральних характеристик та властивостей ідеї потенційного товару є підґрунтям для формування його конкурентоспроможності.

4.2.Технологічний аудит ідеї проекту

Проведемо аудит технологій, за допомогою яких можна реалізувати ідею проекту та визначимо технологічну здійсненність ідеї проекту.

Таблиця 4.3. Технологічна здійсненність ідеї проекту

№	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Реалізація алгоритму	C#	Наявна	Доступна
2	Візуалізація отриманого графа	WPF для десктопу	Наявна	Доступна
		JavaScript/HTML/CSS для вебу	Наявна	Доступна
3	Платформа для зображення структур даних	Додаток на основі ASP.NET Core Framework, який оброблятиме запити користувача	Наявна	Доступна

Продовження таблиці 4.3

Обрана технологія реалізації ідеї проекту: C# + WPF + JavaScript/HTML/CSS + ASP.NET Core

Проаналізувавши інформацію вище можна зробити висновок, шо технологічна реалізація проекту присутня, так як усі технології є доступними. Реалізація вищевказаної бібліотеки та платформи хоч і не наявна зараз, але її розробка не складе складнощів. Тож можемо переходити до аналізу ринкових можливостей запуску стартап-проекту.

4.3. Аналіз ринкових можливостей запуску стартап-проекту

Визначимо ринкові можливості, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту.

Таблиця 4.4.Попередня характеристика потенційного ринку

№	Показники стану ринку	Характеристика
1	Кількість головних гравців, од	5
2	Загальний обсяг продаж, грн./ум.од	2200 в рік
3	Динаміка ринку	Зростає
4	Наявність обмежень для входу	Немає
5	Специфічні вимоги до стандартизації та сертифікації	Практично відсутні
6	Середня норма рентабельності в галузі або по ринку, %	18%

Враховуючи те, що найкраща пропозиція для депозиту в надійному банку складає 16% , а також кількість основних гравців, зростаючу динаміку ринку, відсутні обмеження для входу на ринок, можна зробити висновок, що на даний момент, ринок для входження стартап-продукту є привабливим.

Таблиця 4.5.Характеристика потенційних клієнтів стартап-проекту

№	Потреба, що формує ринок	Цільова аудиторія	Відмінності у поведінці цільових груп клієнтів	Вимоги споживачів до товару
1	Можливість візуалізувати графи простим способом та висока гнучкість при конфігурації.	Програмісти, кінцеві користувачі	Кінцевим користувачам у більшості випадків не потрібна висока гнучкість при конфігурації зовнішнього вигляду кінцевого зображення графа	Коректність отриманого зображення. Зручність і простота у використанні. Також дане рішення повинне бути дуже гнучким

Після визначення потенційних груп клієнтів проведемо аналіз ринкового середовища: складемо таблиці факторів, що сприяють ринковому впровадженню проекту, та факторів, що йому перешкоджають. Фактори в табл. 4.6.Фактори загроз, 4.7.Фактори можливостей подаються в порядку зменшення значущості.

Таблиця 4.6. Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Конкуренти	Наявність конкурентів котрі надають схожі рішення	Розробка унікальних характеристик товару; Подальша підтримка системи
2	Економічний	Закінчення грошей та недостатнє фінансування	Пошук альтернативних джерел фінансування, таких як реклама на платформі, пошук нових інвесторів, дешевших рішень для розгортання системи
3	Низька швидкість візуалізації графа з великою кількістю вузлів і з’єднань	Користувачі втрачають інтерес до бібліотеки через затримки при прокладанні з’єднань між вузлами графа	Оптимізація методу, можливе пожертвування деякими естетичними параметри задля збільшення швидкості

Продовження таблиці 4.6

4	Низька швидкість каналу передачі даних	Користувачі втрачають інтерес до платформи через довгий час відгуку сторінки	Оптимізація платформи, використання кешування даних, оптимізація даних, що передаються з метою зменшення об’єму цих даних

Таблиця 4.7. Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1	Науково-технічний	Покращення і оптимізація методу візуалізації графа	Адаптація існуючого рішення і алгоритмів під нову технологію
2	Новий продукт	Надання нових рішень у сфері	Розробка унікальних характеристик товару
3	Безкоштовність	За рахунок безкоштовності система буде більш привабливою для користувачів	Пошук альтернативних джерел фінансування, таких як реклама в платформі, нових інвесторів, пошук дешевших рішень для розгортання системи для можливості надалі зберігати продукт безкоштовним

Далі наведено результат проведеного аналіз пропозиції де визначаються загальні риси конкуренції на ринку.

Таблиця 4.8. Ступеневий аналіз конкуренції ринку

№	Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1	Тип конкуренції: Чиста конкуренція	Хоча кількість конкурентів не є великою, але поява нових гравців не є ускладненою	Покращення якісних показників, реклама

Продовження таблиці 4.8

2	Рівень конкурентної боротьби: Міжнародний	Фірми-конкуренти знаходяться в інших країнах.	Адаптація продукту як для вітчизняних так і для зарубіжних клієнтів.
3	Галузева ознака: Міжгалузева	Продукт може використовуватися у різних галузях.	Розробка нового функціоналу, контроль коректності даних
4	Конкуренція за видами товарів: Товарно-родова	Види товарів однакові.	Створити продукт, враховуючи сильні і слабкі сторони конкурентів
5	Характер конкурентних переваг: Нецінова	Вдосконалення методу візуалізації.	Постійний розвиток та оновлення, враховування побажань замовників.
6	За інтенсивністю: Марочна	Пропозиція схожого сервісу, спільна цільова аудиторія	Інформування ринку щодо якості використовуваної новаторської бібліотеки, розширення функціоналу

Далі наведено огляд аналізу конкуренції, де проводиться більш детальний аналіз умов конкуренції в галузі. Серед різних факторів зовнішнього середовища, що впливають на фірму, виділяють ключовий - це галузь (чи галузі), в якій фірма веде конкурентну боротьбу. Структура галузі має істотний вплив на формування правил конкурентної боротьби, а також потенційних стратегій фірми. Стан конкуренції в галузі залежить від п'яти основних конкурентних сил (модель п'яти конкурентних сил, розроблена професором Гарвардської школи бізнесу М. Портером).

Таблиця 4.9. Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	GoXam Telerik	Наявність вже існуючих рішеннь	Відсутні	Клієнти можуть бути невдоволенні наявною функціональністю	Товари, які б задовольняли усі вимоги практично відсутні
Висновки	Відсутня монополія зі сторін даних конкурентів, тому вихід на ринок не має додаткових ускладнень.	При розширенні функціональності може стати конкурентом. Невелика кількість необхідних вимог.	Відсутні	Клієнти вносять побажання, що доцільно брати до уваги при розробці продукту.	Відсутні

Проаналізувавши можливості роботи на ринку з огляду на конкурентну ситуацію можна зробити висновок, кожен із існуючих продуктів не задовольняють в повній мірі потреб ринку і тому вихід на даний ринок є можливою і реалізованою задачею.

Також було зроблено висновок щодо характеристик, які повинен мати проект, щоб бути конкурентноспроможним на ринку. Цей висновок був врахований при формулюванні переліку факторів конкурентноспроможності у наступному пункті. На основі аналізу конкуренції, проведеного в табл. 4.9, а

також із урахуванням характеристик ідеї проекту (табл. 4.2), вимог споживачів до товару (табл. 4.5) та факторів маркетингового середовища (табл. 4.6, 4.7) визначається та обґрунтовується перелік факторів конкурентоспроможності. Аналіз оформлюється за табл. 4.10.

Таблиця 4.10. Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспроможності	Обґрунтування
1	Цінова політика	Отримання прибутку здійснюється в основному за рахунок платної підтримки бібліотеки та реклами на платформі, що дає змогу зробити використання додатку безкоштовним
2	Виконання програмного забезпечення у кросплатформеному вигляді	Можливість використання програмного забезпечення на будь-якій платформі.
3	Гнучкість конфігурації зовнішнього вигляду графу	Більше користувачів виберуть бібліотеку для реалізації свої цілей і задач, так як кожен зможе адаптувати зовнішній вигляд під свої потреби
4	Інтуїтивно зрозумілий інтерфейс платформи	Користувачам легше взаємодіяти із платформою

За визначеними факторами конкурентоспроможності проведемо аналіз сильних та слабких сторін стартап проекту.

Таблиця 4.11. Порівняльний аналіз сильних та слабких сторін проекту

№	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні						
			-3	-2	-1	0	+1	+2	+3
1	Цінова політика	18	+						
2	Виконання програмного забезпечення у кросплатформеному вигляді	17			+				
3	Гнучкість конфігурації зовнішнього вигляду графу	16				+			
4	Інтуїтивно зрозумілий інтерфейс платформи	18				+			
6	Локалізація	18					+		

Фінальним етапом ринкового аналізу можливостей впровадження проекту є складання SWOT-аналізу (матриці аналізу сильних (Strength) та слабких (Weak) сторін, загроз (Troubles) та можливостей (Opportunities) (табл. 4.12) на основі виділених ринкових загроз та можливостей, та сильних і слабких сторін (табл. 4.11).

Таблиця 4.12. SWOT аналіз стартап-проекту

Сильні сторони (S):	Слабкі сторони (W):
– Цінова політика – Наявність додаткового функціоналу – Гнучкість конфігурації зовнішнього вигляду графу – Інтуїтивно зрозумілий інтерфейс платформи – Наявність локалізації	– Відносна складність розробки
Можливості (O):	Загрози (T):
– Розширення функціоналу	– Недостатнє фінансування – Видавлення з ринку конкурентами – Зміна потреб користувачів

На основі SWOT-аналізу розробимо альтернативи ринкової поведінки (перелік заходів) для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок.

Визначені альтернативи аналізуватимемо з точки зору строків та ймовірності отримання ресурсів (табл. 4.13.Альтернативи ринкового впровадження).

Таблиця 4.13. Альтернативи ринкового впровадження

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Додавання додаткового функціоналу за певну плату	Ресурси – час, гроші та люди, наявні	5-6 місяців
2	Реклама	Залучення власних коштів для реклами товару	2-3 місяці
3	Зменшення витрат на розробку завдяки використанню безкоштовних рішень	Наявні	3-4 тижні

Обрана альтернатива: Зменшення витрат на розробку завдяки використанню безкоштовних рішень.

4.4.Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку: було проведено опис цільових груп потенційних споживачів (табл. 4.14)

Таблиця 4.14. Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Програмісти	Висока зацікавленість	Попит середній – вище середнього	Вище середньої	Середня складність
2	Користувачі платформи	Середня зацікавленість	Попит середній	Відсутність на початку, подальше збільшення	Складність вище середнього
Які цільові групи обрано: 1 та 2					

Відповідно до проведеного аналізу можна зробити висновок, що обидві групи є підходящими для розповсюдження даного програмного продукту. Так, як простота входу у сегмент з групою 1 простіша в реалізації і, в той же час, група 2 може надати додаткову рекламу продукту та, в перспективі, можливе введення додаткового функціоналу за окрему плату для групи 2. Відповідно до стратегії охоплення ринку збуту товару обрано стратегію масового маркетингу.

Для роботи в обраних сегментах ринку сформуємо базову стратегію розвитку та стратегію конкурентної поведінки.

Таблиця 4.15. Визначення базової стратегії розвитку

Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
Надання функціональності що відсутня у товарів-замінників, підтримка додатку, зворотній зв'язок з клієнтами	Проведення реклами, співпраця з іншими рішеннями у цій сфері, розширення функціональних можливостей системи	Ефективна реклама та співпраця з посередниками, відмітний функціонал рішення	Стратегія диференціації

Наступним кроком обрано стратегію конкурентної поведінки (табл. 4.16). Стратегія конкуренції описується як наступальні чи оборонні (а досить часто — і кооперaційні) дії підприємств, спрямовані на досягнення стійкого становища в галузі, для успішного подолання п’яти чинників конкуренції і, отже, гарантування максимальної віддачі від капіталовкладень. Для того, щоб вижити в сучасних умовах на ринку, фірми, підприємства мають правильно використовувати наступальні та захисні дії у своїх взаємовідносинах із конкурентами.

Таблиця 4.16. Визначення базової стратегії конкурентної поведінки

Чи є проект «першопрохідцем» на ринку	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, які?	Стратегія конкурентної поведінки
Ні	Так, ціль компанії знайти нових споживачів та, частково, забрати існуючих у конкурентів задля задоволення потреб останніх	Компанія частково копіює характеристики продукту конкурента, основна ціль компанії розробка нового унікального функціоналу, з підтримкою основного функціоналу конкурентів м	Стратегія заняття конкурентної ніші

На основі вимог споживачів з обраних сегментів до постачальника (стартап компанії) та до продукту, а також в залежності від обраної базової стратегії розвитку та стратегії конкурентної поведінки розробимо стратегію позиціонування, що полягає у формуванні ринкової позиції (комплексу асоціацій), за яким споживачі мають ідентифікувати торгівельну марку/проект.

Таблиця 4.17. Визначення стратегії похиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту
1	Точність побудованого зображення графа. Висока швидкість візуалізації графа з великою кількістю вузлів та з’єднань, безкоштовність	Стратегія диференціації	Позиція на основі порівняння фірми з товарами конкурентів; Відмінні особливості споживача.	Економія часу; Зручність застосування; Практичність та точність результату

Відповідно до проведеного аналізу можна зробити висновок, що стартап-компанія вибирає як базову стратегію розвитку – стратегію диференціації, як базову стратегію конкурентної поведінки – стратегію заняття нової конкурентної ніші.

4.5.Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримає споживач. Для цього у табл. Таблиця *4.18* потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 4.18. Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1	Візуалізація ациклічного орієнтованого графа	Висока швидкість побудови та задовільність естетичним параметрам	З’єднання розбиті на ортогональні відрізки
2	Зрозумілість та зручність	Естетично приємний дизайн, інтуїтивний інтерфейс	Проведено аналіз існуючих рішень, виділені недоліки в прокладанні з’єднань між вершинами. Вирішено ці проблеми у власному товарі

Розробимо трирівневу маркетингову модель товару: визначимо основну ідею чи вигоду, яку отримує споживач у результаті користування товаром (товар за задумом), наведемо сукупність характеристик та властивостей, матеріальних складових (товар у реальному виконанні) та можливих додаткових складових (послуги, форма оплати), що підвищують цінність товару для споживача (товар із підкріпленням).

Таблиця 4.19. Опис трьох рівнів моделі товару

Рівні товару	Сутність та складові
1. Товар за задумом	Метод працює а режимі реального часу, має високу точність зображення, високу швидкість візуалізації та з’єднання розбиті на ортогональні відрізки

Продовження таблиці 4.19

<table>
<tr><td rowspan="5">2. Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх/Тл/Е/Ор</td></tr>
<tr><td>Візуалізація ациклічного орієнтованого графа

Гнучкість конфігурації зовнішнього вигляду графу

Кросплатформність

Простота</td><td>Нм</td><td>Технологічна</td></tr>
<tr><td colspan="3">Якість: Документація, яка допоможе правильно використовувати бібліотеку</td></tr>
<tr><td colspan="3">Система: бібліотека, яка реалізує метод прокладання з’єднань між вузлами направленого графа</td></tr>
<tr><td colspan="3">Марка: назва компанії</td></tr>
<tr><td rowspan="2">3. Товар із підкріпленням</td><td colspan="3">До продажу: наявна повна документація, безкоштовне використання товару</td></tr>
<tr><td colspan="3">Після продажу: платна підтримка з боку розробників</td></tr>
<tr><td colspan="4">Проект буде захищено від копіювання реєстрацією назви програми, створення заявки на отримання патенту на винахід, щоб уберегти алгоритм роботи від копіювання</td></tr>
</table>

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар, яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів (табл. 4.20). Аналіз проводиться експертним методом.

Таблиця 4.20. Визначення меж встановлення ціни

Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
Замінники відсутні	Більшість аналогів платні, ціна в районі 1000-2000 доларів за ліцензію.	Нижче середнього Середні Вище середнього	На даний момент додаток повністю безкоштовний для користувачів

Наступним кроком визначимо оптимальну систем збуту, в межах якого приймається рішення (табл. 4.21).

Таблиця 4.21. Формування системи збуту

Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
Вибір бібліотек у NuGet Gallery, оплата преміум підписки для активації підтримки від розробників	продаж	Виробник - споживач	NuGet Gallery, Офіційний сайт виробника із сторінкою прайсингу та придбання

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів (табл. 4.22).

Таблиця 4.22. Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуютьс я цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомленн я	Концепція рекламного звернення
1	Придбання ліцензії шляхом отримання унікальног о ключа	Інтернет	Адаптивність Конфіденційніст ь Простота	Показати переваги сервісу, у тому числі і перед конкурентам и	Демо-ролик застосуванн я Таргетингов а реклама

4.6.Висновки до розділу

Під час аналізу було виявлено, що ринок на даний момент зростає, наявний попит на системи такого плану та відносно невеликий рівень конкуренції та майже відсутні обмеження для входу на ринок. Враховуючи ці фактори можна

зробити висновок, що на даний момент існує можливість ринкової комерціалізації проекту.

На даний момент існує декілька схожих рішень, але жодне із них не є монополістом на ринку. Так, як проект має переваги над уже існуючими системами, ринок для входження стартап-продукту є привабливим.

Альтернативним варіантом впровадження є розробка специфічного функціоналу за додаткові кошти.

Проаналізувавши фактори, можна дійти до висновку, що подальша реалізація проекту є доцільною.

ЗАГАЛЬНІ ВИСНОВКИ

В ході виконання магістерської дисертації було досліджено методи візуалізації ациклічних орієнтованих графів. Було проведено порівняльний аналіз текстового та візуального програмування, наведено базову термінологію теорії графів. Також було проведено аналіз існуючих алгоритмів візуалізації графів. Було розглянуто особливості кожного алгоритму, наведено їх переваги та недоліки. Досліджено проблеми, які виникають при візуалізації графів та існуючі підходи до візуалізації графів. На основі отриманих знань обґрунтовано актуальність оптимізації швидкості знаходження найкоротших шляхів між вузлами направленого графа за рахунок створення власного методу.

Поставлено задачі, основною метою яких є створення алгоритму візуалізації з'єднань графа, який буде виконувати поставлену мету та задовольнятиме висунутим критеріям оптимальності.

Проведено опис запропонованого методу знаходження найкоротших шляхів та прокладання з’єднань у орієнтованому ациклічному графі. Аргументовано для чого потрібна кожна складова методу та що досягається в результаті її виконання. Сформульовано етапи із яких складається запропонований метод та створено алгоритм візуалізації з'єднань графа.

Обрано підходи та технології для імплементації методу. Реалізовано алгоритм у вигляді бібліотеки для відображення графа. Проведено експерименти та доведено ефективність реалізованої бібліотека та власне методу.

Проведено аналіз ринку та можливостей виходу даного проекту на цей ринок. Виявлено, що існують перспективи виходу на ринок та існує доцільність подальшого розвитку проекту. Також розроблені ринкова стратегія та маркетингова програма стартап-проекту.

Результатом виконання магістерської дисертації є запропонований метод знаходження найкоротших шляхів та прокладання з’єднань у орієнтованому ациклічному графі. Створена бібліотека, яка реалізує алгоритм та може бути використана іншими розробниками для відображення графа.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Visual programming language [Електронний ресурс]: веб-сторінка онлайн довідника Вікіпедія. – Режим доступу https://en.wikipedia.org/wiki/ Visual_programming_language.
2. Выбор между визуальным и текстовым программированием для детей [Електронний ресурс]: веб-ресурс habr. – Режим доступу https://habr.com/post/400495/.
3. Graph [Електронний ресурс]: веб-сторінка онлайн довідника Вікіпедія. – Режим доступу https://en.wikipedia.org/wiki/Graph(discrete_mathematics).
4. Directed acyclic graph [Електронний ресурс]: веб-сторінка онлайн довідника Вікіпедія. – Режим доступу https://en.wikipedia.org/wiki/Directed_acyclic_graph
5. Circle graph [Електронний ресурс]: веб-сторінка онлайн довідника Вікіпедія. – Режим доступу https://en.wikipedia.org/wiki/Circle_graph
6. Arc diagram [Електронний ресурс]: веб-сторінка онлайн довідника Вікіпедія. – Режим доступу https://en.wikipedia.org/wiki/Arc_diagram
7. Heer Jeffrey. A tour through the visualization zoo / Heer Jeffrey; Bostock Michael; Ogievetsky Vadim – Communications of the ACM, 2010. – С. 59-67.
8. Force-directed graph drawing [Електронний ресурс]: веб-сторінка онлайн довідника Вікіпедія. – Режим доступу https://en.wikipedia.org/wiki/Force-directed_graph_drawing
9. Візуалізація графів [Електронний ресурс]: веб-ресурс science-community. – Режим доступу https://www.science-community.org/en/node/5582
10. Построение изображений ациклических ориентированных графов [Електронний ресурс]: веб-ресурс rain.ifmo.ru. – Режим доступу http://rain.ifmo.ru/cat/view.php/theory/graph-coloring-layout/dag-planarization-2007
11. Dijkstra's algorithm [Електронний ресурс]: веб-сторінка онлайн довідника Вікіпедія. – Режим доступу https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

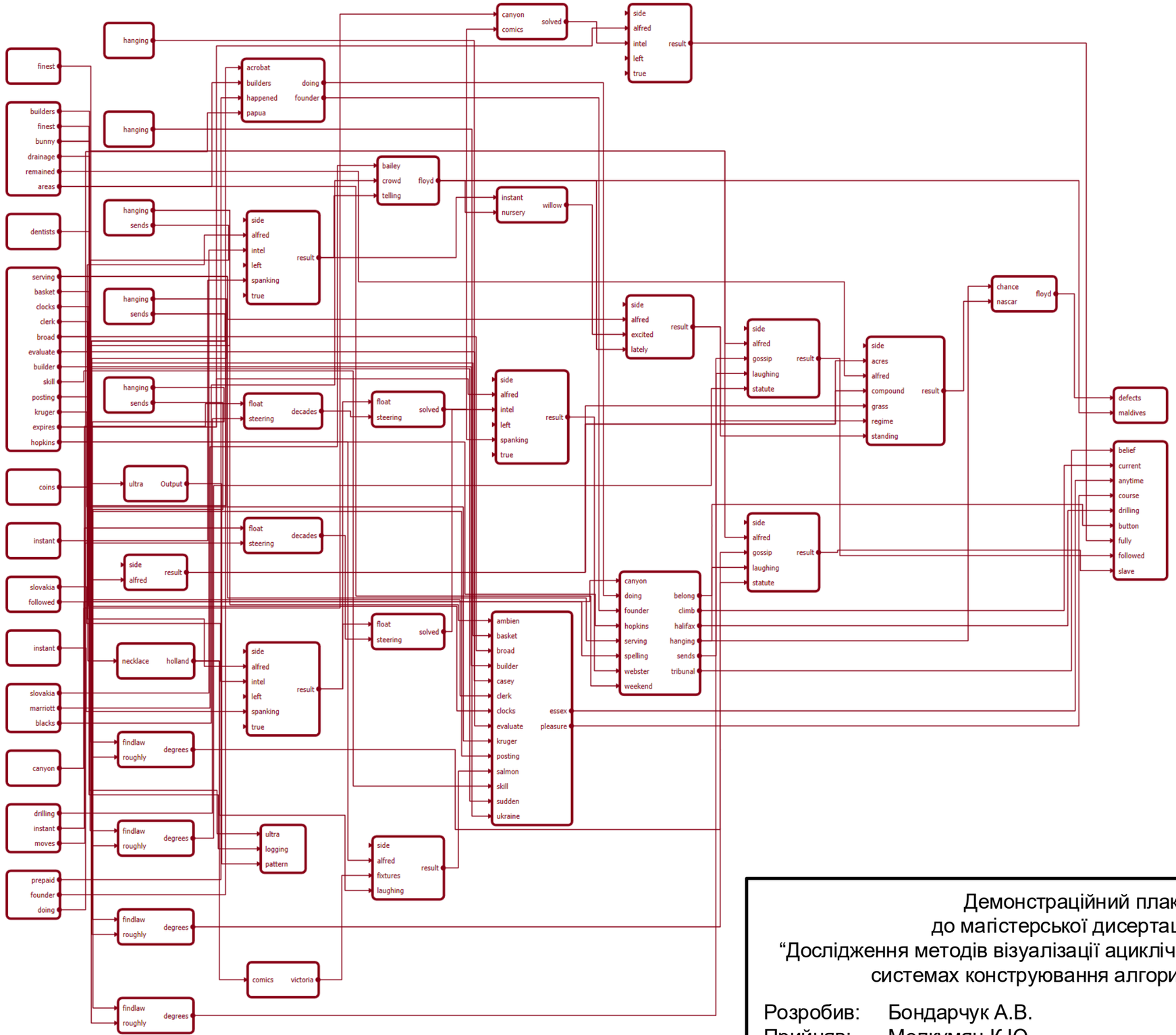
12. A* search algorithm [Електронний ресурс]: веб-сторінка онлайн довідника Вікіпедія. – Режим доступу https://en.wikipedia.org/wiki/A*_search_algorithm

ДОДАТКИ

ДОДАТОК А

Результат виконання методу

Результат виконання методу



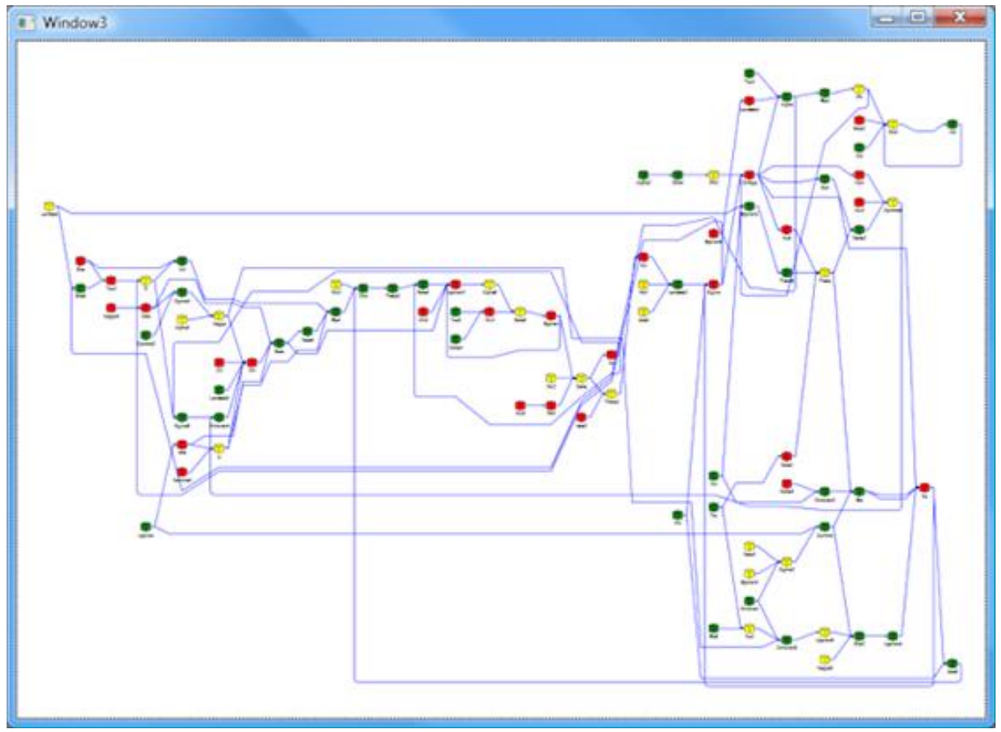
Демонстраційний плакат №1
до магістерської дисертації на тему
“Дослідження методів візуалізації ациклічних орієнтованих графів у системах конструювання алгоритмів обчислень”

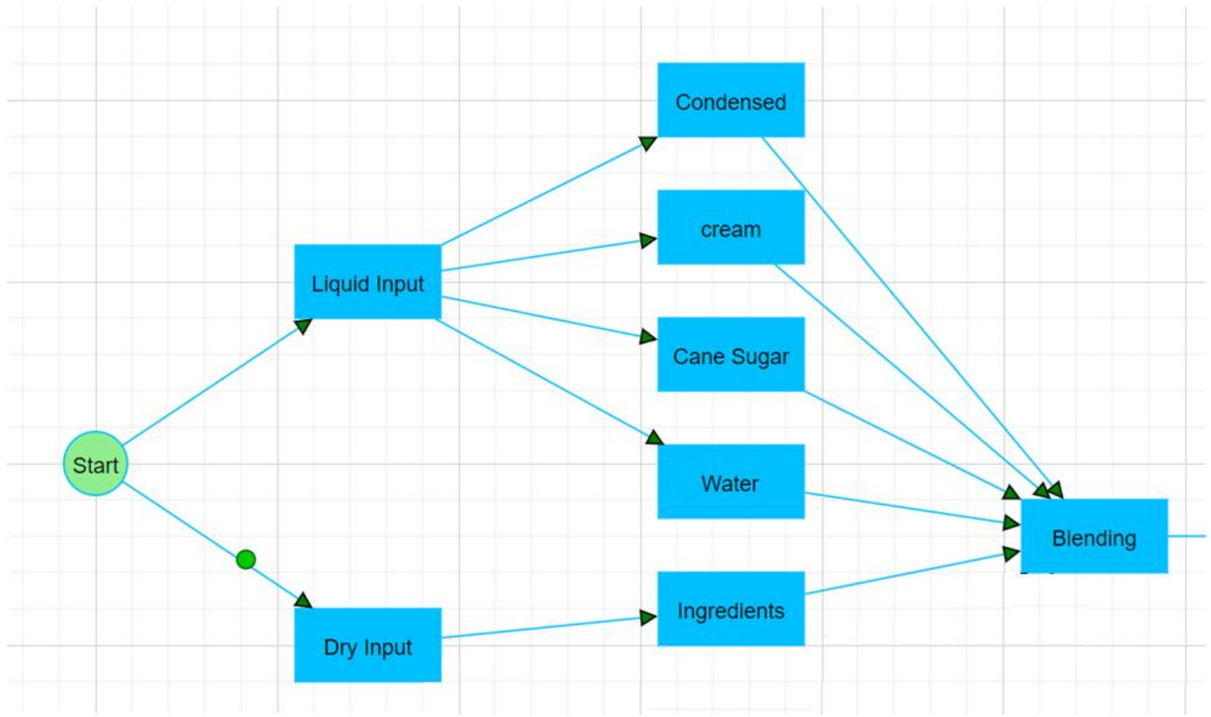
Розробив: Бондарчук А.В.
Прийняв: Мелкумян К.Ю.

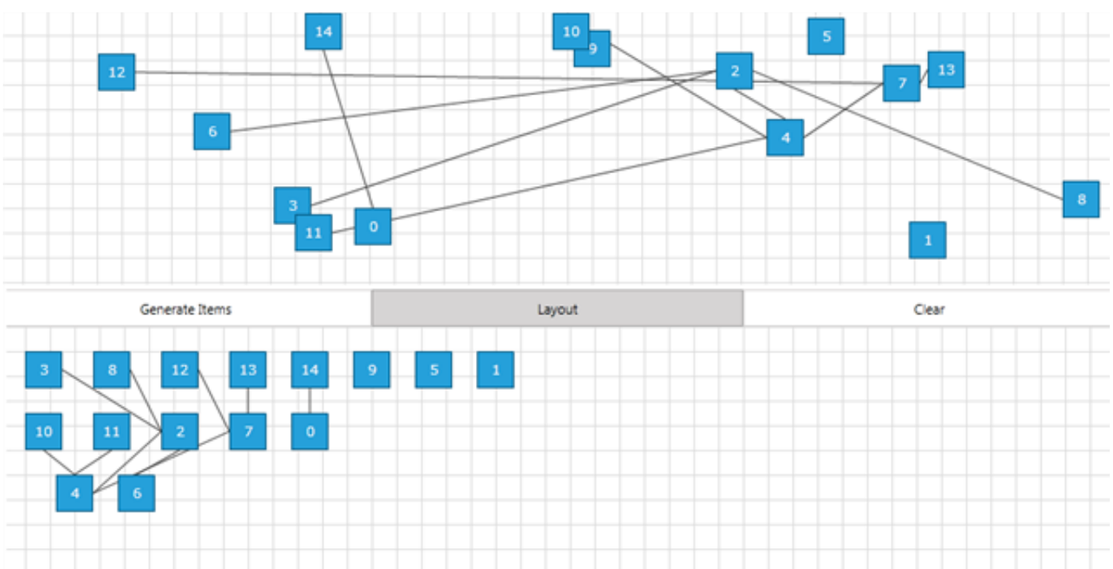
ДОДАТОК Б

Скріншоти роботи існуючих рішень

Скріншоти роботи існуючих рішень







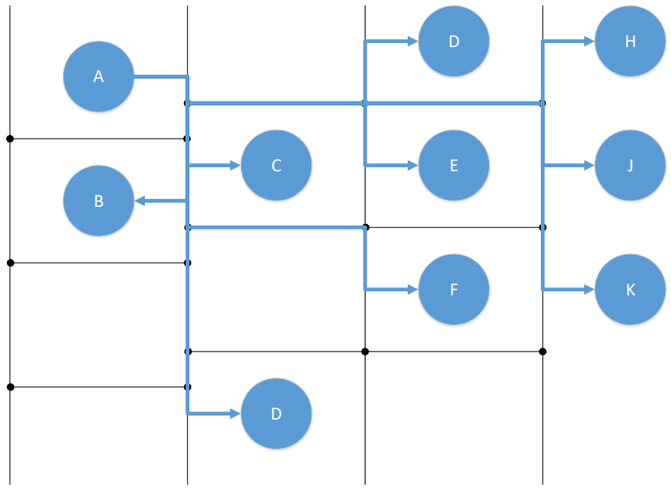
Демонстраційний плакат №2
до магістерської дисертації на тему
“Дослідження методів візуалізації ациклічних орієнтованих графів у системах конструювання алгоритмів обчислень”

Розробив: Бондарчук А.В.
Прийняв: Мелкумян К.Ю.

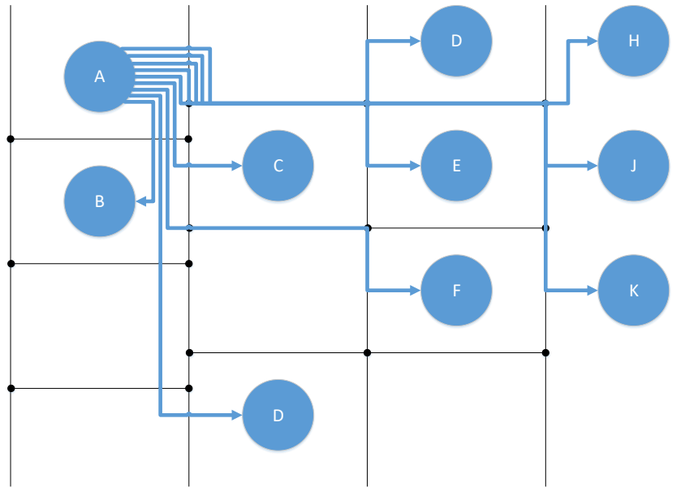
ДОДАТОК В

Розгортання шин у з’єднання

Розгортання шин у з’єднання

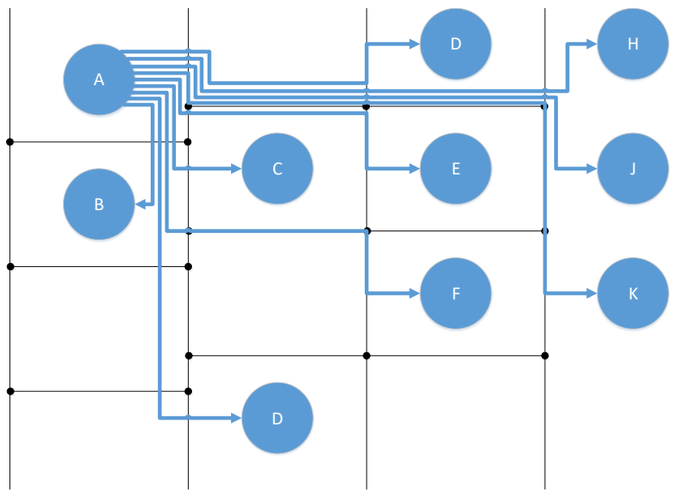


1)

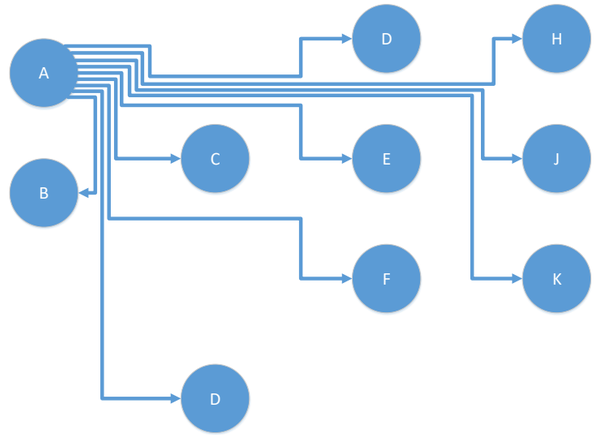


2)

3)



4)



5)

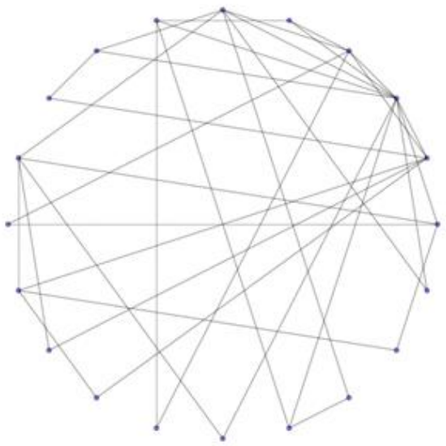
Демонстраційний плакат №3
до магістерської дисертації на тему
“Дослідження методів візуалізації ациклічних орієнтованих графів у системах конструювання алгоритмів обчислень”

Розробив: Бондарчук А.В.
Прийняв: Мелкумян К.Ю.

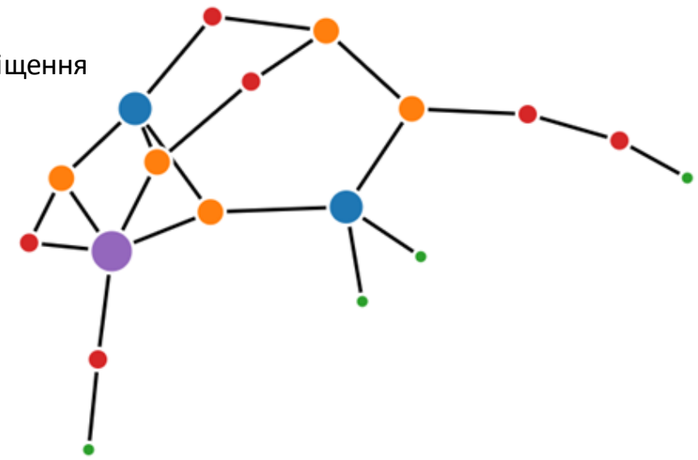
ДОДАТОК Г

Алгоритми візуалізації графів

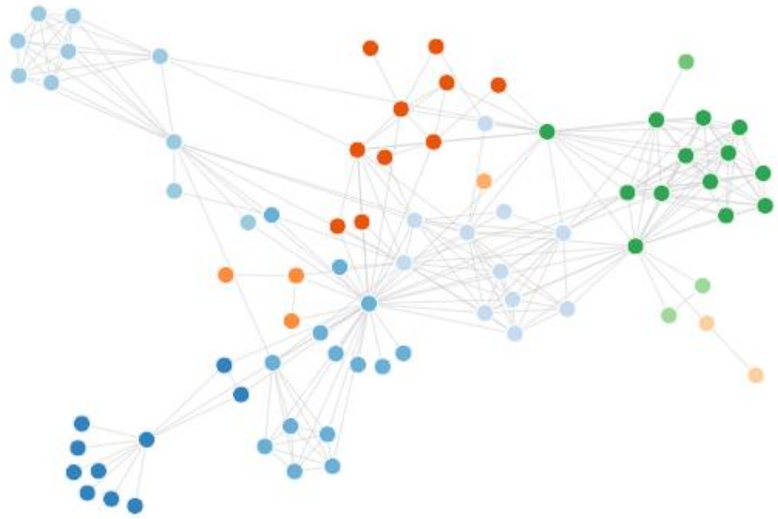
Алгоритми візуалізації графів



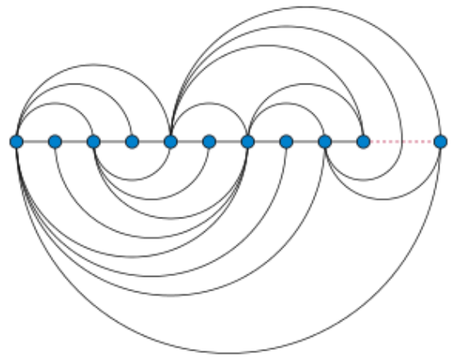
Алгоритм кругового розміщення



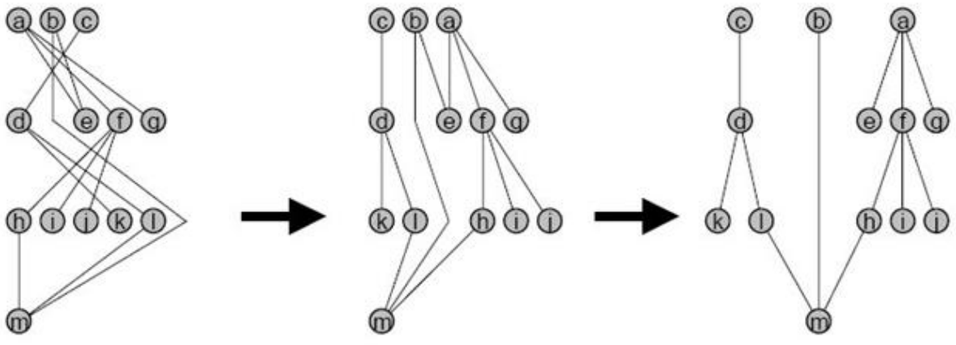
Алгоритм випадкового розміщення



Алгоритм «Розміщення під дією сили»



Алгоритм дугового розміщення



Алгоритм Сугіями

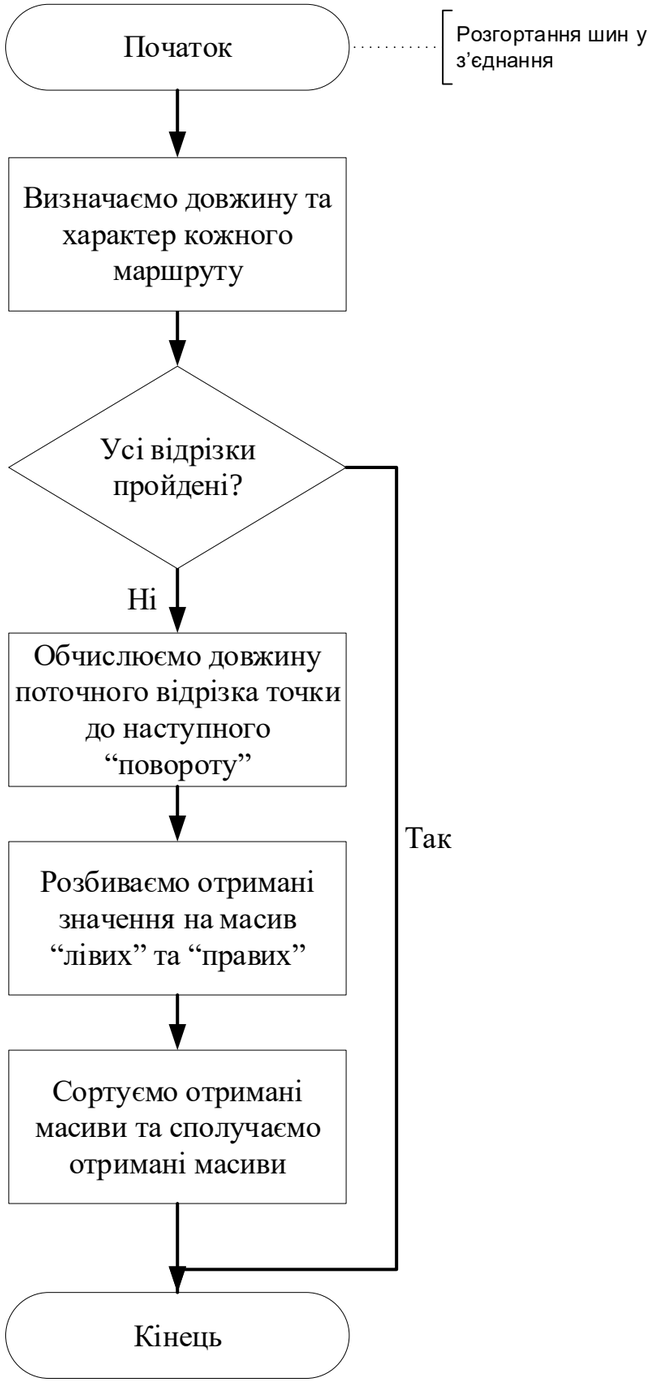
Демонстраційний плакат №4
до магістерської дисертації на тему
“Дослідження методів візуалізації ациклічних орієнтованих графів у системах конструювання алгоритмів обчислень”

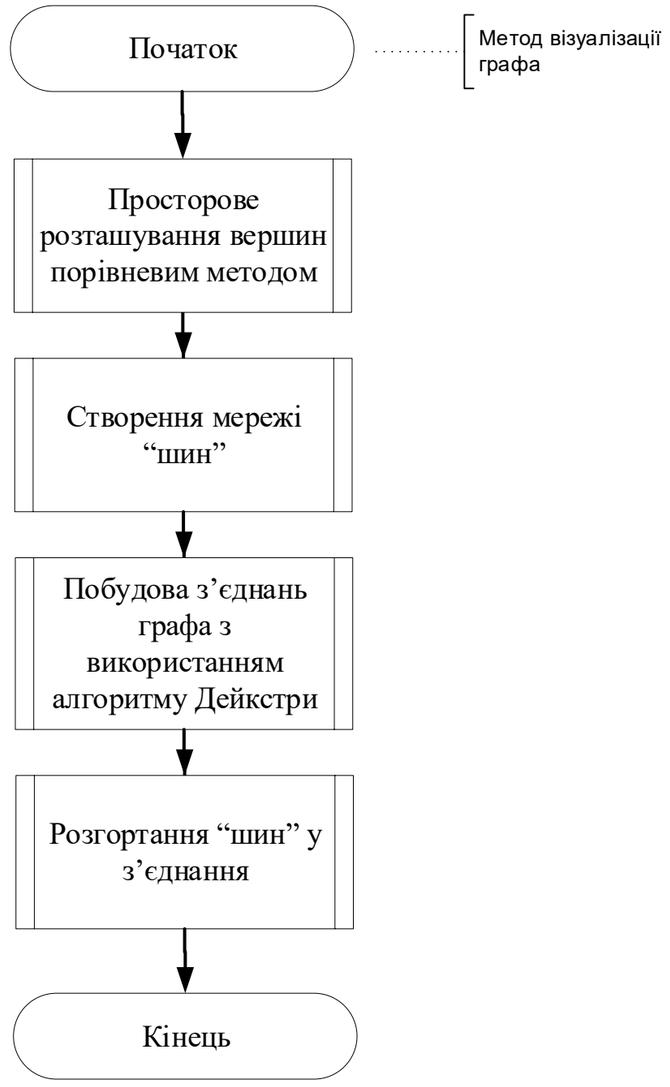
Розробив: Бондарчук А.В.
Прийняв: Мелкумян К.Ю.

ДОДАТОК Ґ

Блок-схема запропонованого методу

Блок-схема запропонованого методу





Демонстраційний плакат №5
до магістерської дисертації на тему
“Дослідження методів візуалізації ациклічних орієнтованих графів у системах конструювання алгоритмів обчислень”

Розробив: Бондарчук А.В.
Прийняв: Мелкумян К.Ю.